UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI
UNDERGRADUATE SCHOOL

Research and Development

BACHELOR THESIS

by

An Minh Tri

BI12-447

Information and Communication Technology

Title:

Modification of LipNet on Smaller Data Scale and Application Deployment

External Supervisor: Prof. Nguyen Duc Dung – IOIT/VAS

Internal Supervisor: Dr. Kieu Quoc Viet – ICT Lab

Hanoi. July 2024

Acknowledge

Firstly, I would like to thank my research supervisor, Dr Nguyen Duc Dung, for granting me the permission to work online so that I can help my family matters in Saigon. He also granted me the permission to access the IOIT server, gave me invaluable guidance and feedback during this internship. Secondly, I would like to thank my friends in the Data Science major: Son Nguyen and Quoc Nguyen, for their helps to finish the paperwork procedure for this internship while I was away. Lastly, I would like to express my deep and sincere gratitude to my family, my professors in USTH and all the friends I made along the way throughout my 3-year journey for supporting me.

(An Minh Tri) Hanoi,

July 2024

Table of Contents

List of Acronyms

API: Application Programming Interface

Bi-GRU: Bidirectional Gated Recurrent Unit

CER: Character Error Rate

CNN: Convolutional Neural Network

Conv3D: 3D Convolutional Neural Network

CTC: Connectionist Temporal Classification

DL: Deep Learning

GRU: Gated Recurrent Unit

LSTM: Long Short-Term Memory

ML: Machine Learning

ReLu: Rectified Linear Unit

RGB: Red Green Blue

RNN: Recurrent Neural Network

SD3D: Spatial Dropout 3D

STCNN: Spatiotemporal Convolutional Neural Network

WER: Word Error Rate

List of Figures

List of Tables

Abstract

Lip reading is a method of communication by interpreting the movements of the lips based on visual cues. People built lip movements detection systems based on how human would think when lip reading, particularly for improving communication between the hearing-impaired people and others. Lip reading translation systems can also replace speech recognition systems in noisy environments. Throughout history, machine learning (ML), especially deep learning (DL) has brought many state-of-the-art models and results regarding lip reading detection problems. LipNet, an end-to-end sentence-level lip reading model, set state-of-the-art performance on the GRID dataset but it requires extensive dataset and significant computational resources for training. In this thesis, we will explore the methodology to modify the LipNet model to operate effectively on a smaller data scale, suitable for our limited computational power and time. Our work focuses on optimizing LipNet to function efficiently with reduced data, getting decent results such as 11.3% WER and 3.8% CER on overlapped speaker test set, and 23.8% WER and 10.9% CER on unseen speaker test set. The modified model is then deployed as a desktop application, aimed at providing a usable lip reading translation tool.

Keywords: *Computer vision, Natural language processing, Model optimization, CTC, LipNet, Desktop Application.*

Chapter 1

Introduction

1.1 Overview

Lip reading technology has made significant advancements in recent years, particularly from deep learning models. These models have shown impressive performance on large-scale datasets, accurately interpreting speech by visually analyzing lip movements. But there are not many applications or deployment of such models, therefore we decided to implement a DL model to deploy our lip reading translation application. We chose LipNet as it is one of the first end-to-end models for interpreting lip shapes and movements, achieving state-of-the-art performance on many datasets. However, we faced a challenge that was not fully addressed by the paper, namely the training process is very time consuming and requires strong computational power.

By enhancing the performance of LipNet on smaller data scale, we made the technology more accessible to us who cannot afford the time and computational power to train the model on the original data scale.

To modify LipNet and solve our data problems, it is crucial to understand the context regarding visual speech recognition and the approach LipNet took in their original paper. There are 5 different approaches regarding the visual speech recognition problem[2]:

- **Digits:** This approach focuses on recognizing individual digits (0-9) from visual speech cues. The recognition task is the simplest compared to other approaches.
- **Alphabet:** This approach deals with recognizing individual letters of the alphabet (a-z) from visual speech cues. It is more challenging than digit recognition, as the visual differences between certain letters can be subtle such as "b" and "p".
- **Words:** This approach aims to recognize individual words from visual speech cues. It is more complex than digit or alphabet recognition, as words can have

varying lengths and pronunciations.

- **Phrases:** This approach deals with recognizing short, multi-word phrases from visual speech cues. Phrase recognition is more complex than word recognition, as it requires understanding the context and relationships between the words.
- **Sentences:** This approach is the most complex, as it involves recognizing entire sentences from visual and contextual information within the spoken language such as grammar and semantics. This is the approach LipNet took in their original paper, therefore will also be our approach in this thesis.

Replicating the structure and results from a published work on a visual speech recognition task can be highly challenging due to several influential factors. Firstly, the size and composition of the dataset used in the original study is severely larger than our data scale. This discrepancy can lead to variations in the performance of the models, as the dataset plays a crucial role in training and evaluating the systems. Additionally, the data preprocessing steps and hyperparameters employed may differ due to the first reason. Furthermore, the model structures and some hyperparameters described in the paper are either ambiguous or incomplete, making it difficult to replicate the exact architecture and training procedure used in the original study. Overcoming these challenges requires a significant amount of experimentation finding the fitting techniques and structures for the best results in our capacity.

Our goal is to achieve decent results (roughly 20% WER on unseen speakers would be considered decent) to deploy our lip reading translation application. Below is our framework for this project:

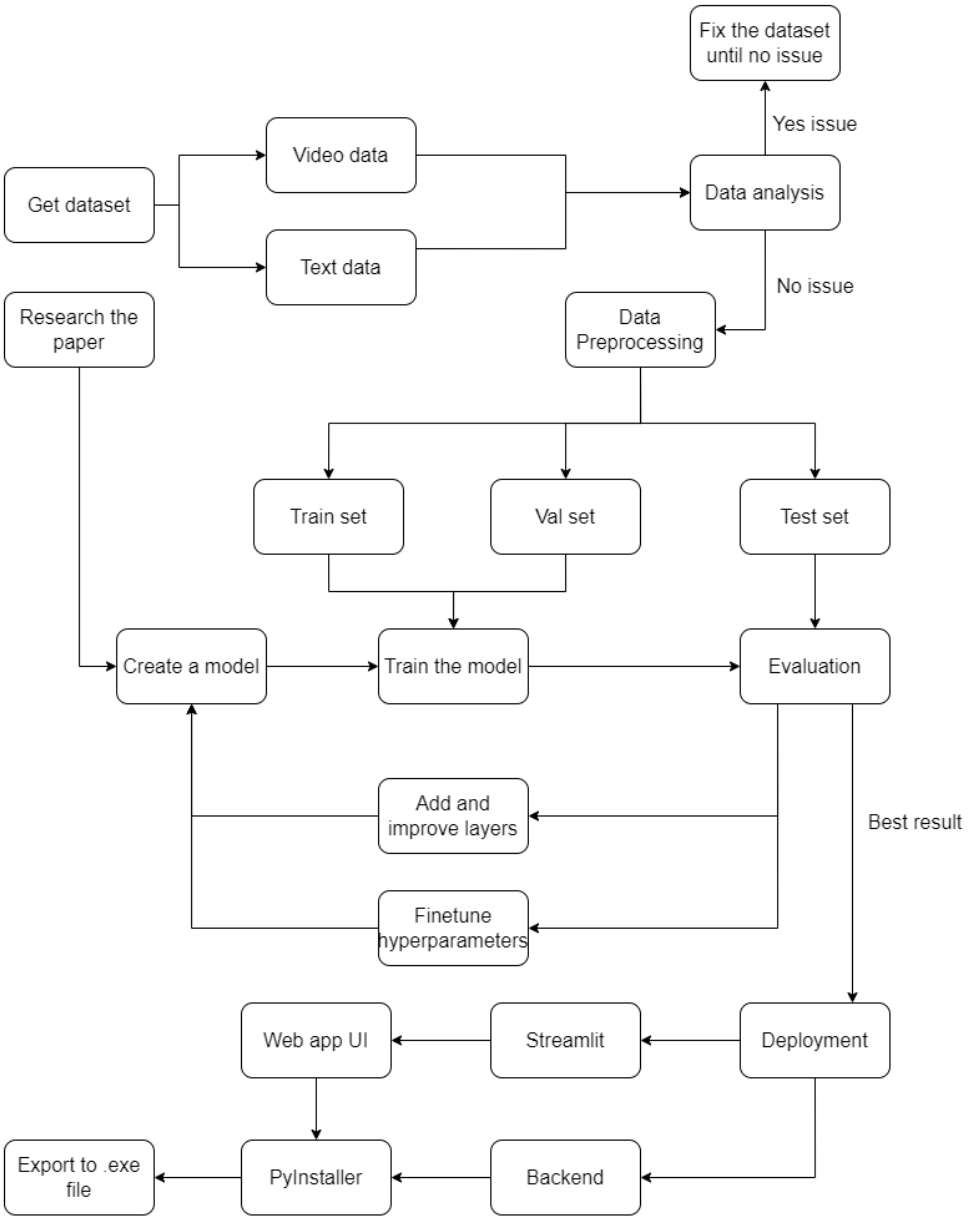


Figure 1. 1 – Overall framework of this project

1.2 Related Works

The field of automatic lip-reading has seen a variety of approaches over the years. Early methods relied heavily on hand-crafted features and traditional machine learning techniques. These methods focused on extracting visual features such as lip contours, shapes, and motion patterns, and then used techniques like Hidden Markov Models[3] or Support Vector Machines for classification[4]. For instance, Potamianos et al. (2003)[6] explored the use of appearance-based features combined with HMMs to perform lip-reading, achieving modest success on small datasets.

With the rise of deep learning, more sophisticated models have been developed. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) became the standard for feature extraction and sequence modeling, respectively. Multiple deep learning models, such as LipNet[1], Recurrent Temporal Multimodal Restricted Boltzmann Machines[5] have demonstrated significant improvements in performance and scalability.

Transformer architectures have brought advancements recently in lip-reading. These models, known for their ability to handle long-range dependencies and parallelize training, have shown promise in various sequence-to-sequence tasks. Deep Lip Reading: a comparison of models and an online application (2018) by Xu et al.[7], a Transformer-based model designed specifically for lip-reading, leveraging the self-attention mechanism to capture difficult lip movement patterns. The study compared different architectures and demonstrated the effectiveness of transformers in this domain.

Table 1. 1 – Existing methods and datasets on lip reading

Method	Dataset	Data Size	Output
Potamianos et al.	CUAVE	Thousands of utterances	Digits and phrases
Navdeep Jaitly et al.	AVLetters	3,080 spoken letters	Phrases
Stavros Petridis et al.	LRS2-BBC	45839 sentences	Alphabet
Chung et al.	LRS3-TED	400 hours of video	Words
Xu et al.	LRS2	Over 2-million-word instances	Sentences
Assael et al. (LipNet)	GRID corpus	28775 sentences	Sentences
Our Model	GRID corpus	2799 sentences	Sentences

1.3 Motivation

One of the primary motivations behind this research is to provide an alternative mean of communication for the deaf community. In noisy environments, traditional speech-to-text systems often struggle to accurately transcribe spoken language due to the interference from background noise. Lip reading, on the other hand, relies solely on visual cues, making it a more reliable form of communication in such settings. Additionally, writing can be time-consuming and sign language requires specific knowledge that not everyone possesses, the communication aid this project provided only requires lip movements, therefore people can efficiently communicate without the need for writing or sign language. On the technical side, we wanted to study more on how a visual speech recognition model works, and how we can modify that model for our use.

1.4 Objectives

In this internship, we create an end-to-end system optimizing LipNet on smaller data scale that utilizes OpenCV and DL techniques to recognize visual speech. Our goal in this internship is to achieve roughly 20%-30% WER on unseen data and 10%-20% WER on overlapped data. The results from our best DL model will be compared to other published works on both WER and CER for seen and unseen speakers. Our model will be deployed as a desktop web application that can translate lip movements into English text with audio support using Streamlit and PyInstaller.

1.5 Thesis Structures

In this section, we will summarize the content of each chapter thoroughly:

- **Chapter 2 (Background)** represents the techniques we used for data preprocessing, the layers implemented into the model architecture and evaluation metrics.
- **Chapter 3 (Materials and Methods)** gives the information on the dataset and the LipNet model, the methods we used to preprocess the data and modify the model.
- **Chapter 4 (Experiment and Results Analysis)** how we setup the experiment and evaluate our model with other studies.
- **Chapter 5 (Deployment)** how we deploy our model as a desktop application that can translate lip movements.
- **Chapter 6 (Conclusion and Future Works)** summarizes what we have learned and done in this internship, proposes future research and product directions.

Chapter 2

Background

2.1 Data Preprocessing Techniques

2.1.1 Face Detection

Face detection is a common and traditional data preprocessing technique when it comes to problems related to facial analysis and recognition such as lip reading, emotion detection, facial expression classification, ... For this project, we utilized Google's MediaPipe Face Detection[*] model application programming interface (API) to detect the location and region of the face. The face detection model takes an image or a frame of a video as input and will output a bounding box (x, y, w, h) around the detected face, we then take the output and crop out the irrelevant area to prevent our lip reading model from learning unnecessary features and reduce data size.

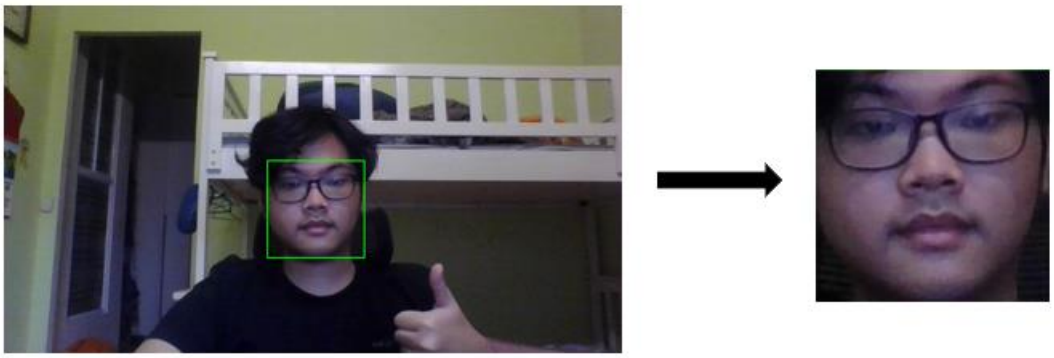


Figure 2. 1 – Face detection bounding box and after cropped

2.1.2 Grayscale

Grayscale is a technique used to convert a red, green, blue (RGB) color image into a grayscale image, which contains only shades of gray, ranging from black to white. This process reduces the complexity of the image data by removing color information and retaining only the brightness information. The formula [8] is defined by:

[*] Google AI Developer Team. MediaPipe Face Detection. Guide and API link: https://ai.google.dev/edge/mediapipe/solutions/vision/face_detector

$$\text{Gray} = 0.2989 \cdot R + 0.5870 \cdot G + 0.1140 \cdot B$$

Where:

- R is the pixel intensity of red.
- G is the pixel intensity of green.
- B is the pixel intensity of blue.

The technique was mainly used to reduce dataset size and complexity while still maintain the edges and contours features, therefore reducing the time needed to load, process data and training. It will need further experiment to see if whether grayscale data affects the final results or not, we will go in detail on chapter 4 (Experiment).

2.1.3 Z-score Normalization

Z-score Normalization[9] or standard score normalization is a type of standardization that involves transforming the whole dataset to have a zero mean and unit variance. The process will take a frame of a video as input, calculate the mean and standard deviation throughout all the pixels in that frame, then run each pixel one by one through two formulas.

Zero Mean: the mean of the pixel values is subtracted from each pixel value. This centers the data around zero.

Unit Variance: After centering the data, each pixel value is divided by the standard deviation of the pixel values. This scales the data to have a standard deviation of 1.

$$z_i = \frac{x_i - \mu}{\sigma}$$

Where:

- x_i is the original pixel value.
- μ is the mean of all the pixels in that frame.
- σ is the standard deviation of all the pixels in that frame.

This process works on both RGB and grayscale image/frame as X can be a single number or an array with 3 elements, for RGB image/frame the mean and standard deviation will be corresponding to their channels.

2.2 Implemented Model Layers

2.2.1 Spatiotemporal Convolutional Neural Network

Spatiotemporal Convolutional Neural Network (STCNN) extracts spatiotemporal features from the input frames. It consists of layers of 3D convolutional neural networks (Conv3D) connect with activation functions and spatial pooling layers. In this project, we use Rectified Linear Unit (ReLu) as activation functions and spatial max-pooling layers for our STCNNs.

Conv3D layers: perform spatiotemporal features extraction by applying convolutional filters over three dimensions: height, width, and time (or frames in this case). A basic 3D convolution layer from C channels to C' channels (without a bias and with unit stride) computes as below:

$$Y(i,j,k) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \sum_{p=0}^{P-1} X(i+m, j+n, k+p) \cdot W(m,n,p)$$

Where:

- Y(i, j, k) is the output feature map.
- X is the input volume.
- W is the 3D kernel (filter).
- i, j, k are the spatial coordinates of the output volume.
- m, n, p are the coordinates within the kernel dimensions.
- M, N, P are the dimensions of the kernel.

ReLu: is an activation function, it introduces non-linearity into the model, allowing it to learn more complex patterns.

$$f(x) = \max(0, x)$$

Spatial max-pooling layers: reduce the dimensionality while preserving the most important features. These layers down sample the input by taking the maximum value over small spatial and temporal regions.

$$y_{i,j,k} = \max_{m,n,p} \left(x_{i+m, j+n, k+p} \right)$$

Where:

- x is the input 3D feature map.
- y is the output feature map after max-pooling.
- m, n, and p are the indices that iterate over the spatial neighborhood defined by the pooling kernel.
- i, j, and k are the indices of the output feature map y.

2.2.2 Gated Recurrent Unit

Gated Recurrent Unit (GRU)[10] is a type of RNN that is designed to handle sequence data similar to Long Short-Term Memory (LSTM) and is known for its ability to capture long-term dependencies. It uses gating mechanisms (update gate and reset gate) that help control the flow of information and mitigate the vanishing gradient problem. The standard GRU formulas go as below:

- **Update Gate:**

$$z_t = \sigma(W_z \cdot x_t + U_z \cdot h_{t-1} + b_z)$$

- **Reset Gate:**

$$r_t = \sigma(W_r \cdot x_t + U_r \cdot h_{t-1} + b_r)$$

- **Candidate Activation:**

$$\tilde{h}_t = \tanh(W_h \cdot x_t + r_t \odot (U_h \cdot h_{t-1}) + b_h)$$

- **New Hidden State:**

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

Where:

- x_t is the input at time step t.
- h_{t-1} is the hidden state from the previous time step.
- W_z, W_r, W_h are the weight matrices for the update gate, reset gate, and candidate activation, respectively.
- U_z, U_r, U_h are the recurrent weight matrices.
- b_z, b_r , b_h are the bias vectors.
- σ is the sigmoid activation function.

- tanh is the hyperbolic tangent activation function.
- ⊙ denotes the element-wise multiplication.

We implemented Bidirectional GRU (Bi-GRU), a variant of GRU, for this project as it can process the input sequence in both forward and backward directions, allowing the model to have both past and future context at each time step.

2.2.3 Linear/Dense

A Linear or Dense layer is a fully connected layer, meaning each neuron in the layer is connected to every neuron in the previous layer. In the context of this project, the Dense layer is applied as the output of the model and will predict the character with the highest probability using the Softmax activation function at each time step.

Softmax: is an activation function commonly used in the final layer of a classification model. It converts raw output scores (logits) from the network into probabilities so that they sum to 1:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Where:

- z is the input vector.
- z_i is the i-th element of the input vector z.
- K is the number of classes (elements in the vector z).

2.2.4 Connectionist Temporal Classification

Connectionist Temporal Classification (CTC)[11] is a loss function designed for sequence-to-sequence problems where the alignment between the input and output sequences is unknown. This is particularly useful in tasks like speech recognition and lip reading, where the length of the input sequence (frames of video or audio) can vary significantly from the length of the output sequence (text). It evaluates all possible alignments between the input and output sequences and calculates the probability of the correct output sequence by summing the probabilities of all valid alignments, this allows for the possibility of repeating characters and insertions of a special "blank" token denoted as "-". The goal during training is to maximize this probability and minimize the loss function.

The probability of an alignment path is given by:

$$p(\pi|\mathbf{x}) = \prod_{t=1}^{T} y_{\pi_t}^{(t)}$$

The probability of the target sequence is the sum over all possible alignment paths:

$$p(\mathbf{y}|\mathbf{x}) = \sum_{\pi \in \mathcal{B}^{-1}(\mathbf{y})} p(\pi|\mathbf{x})$$

The CTC loss is then the negative log probability of the target sequence:

$$\mathcal{L}_{\text{CTC}} = -\log(p(\mathbf{y}|\mathbf{x}))$$

Where:

- **x** is the input sequence.
- π is a valid alignment.
- T is the length of the input sequence.
- $y_{\pi t}^{(t)}$ is the probability of the label π_t at time step t.
- y is the target output sequence.
- B is a mapping function that removes blanks and repeated characters.

2.2.5 Batch Normalization

Batch Normalization[12] is a technique used to improve the training of deep neural networks by standardizing the inputs to each layer. It normalizes the input features by scaling them to have zero mean and unit variance within each mini-batch similar to the Z-score Normalization technique. After normalization, batch normalization introduces learnable parameters to scale and shift the normalized output:

Compute the mean and variance for each mini batch:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \qquad\qquad \sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

Normalize the input:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

Scale and shift:

$$y_i = \gamma \hat{x}_i + \beta$$

Where:

- x_i is the input value.
- μ_B is the mean of the batch.
- σ_B^2 is the variance of the batch.
- $\hat{x}_i$ is the normalized input.
- γ is a learned scale parameter.
- β is a learned shift parameter.
- ϵ is a small constant added for numerical stability.

Batch normalization can be applied to any layer of the neural network, often used after convolutional or fully connected layers and before the activation function.

2.2.6 Dropout

Dropout[13] is a regularization technique used in neural networks to prevent overfitting. It works by randomly "dropping out" (setting to zero) a fraction of the neurons during the training process. This helps to ensure that the network does not become too reliant on specific neurons. During training, dropout is applied to the neurons but during inference (testing/validation), no neurons are dropped, but the weights are scaled down to account for the dropped neurons during training:

Dropping Neurons During Training:

$$h_{\text{dropped}} = h \odot m$$

Scaling During Inference:

$$h_{\text{inference}} = h \cdot (1 - p)$$

Where:

- h is input of the layer
- m is a mask generated from a Bernoulli distribution with probability 1−p.
- p is the dropout rate.
- $\odot$ denotes the element-wise multiplication.

2.2.7 Spatial Dropout 3D

Spatial Dropout 3D (SD3D) is a variant of dropout designed specifically for 3D convolutional layers. It helps to prevent overfitting by randomly setting entire feature maps to zero rather than individual elements while maintains the spatial relationships within the remaining feature maps.

2.2.8 Saliency Map*

Saliency Map[14] is not a layer in a model, but a visualization technique used to highlight the most important regions of an input image that a neural network focuses on while making a prediction. It helps to understand which parts of the input are most relevant for the decision-making process of the model. The saliency map is often computed based on the gradients of the output with respect to the input image. These gradients indicate how much a small change in each pixel of the input image would affect the output prediction. To compute the saliency map, we backpropagate the gradients from the output layer to the input layer:

$$S = \left| \frac{\partial O}{\partial I} \right|$$

Where:

- S is the saliency map.
- O is the output of the neural network.
- I is the input to the neural network.

2.3 Evaluation Metrics

2.3.1 Word Error Rate

Word Error Rate (WER) is a metric used to evaluate the performance of automatic speech recognition and other sequence-to-sequence models including lip-reading systems. It measures the percentage of words that are incorrect in the predicted transcription compared to the reference (ground truth) transcription. WER is calculated as:

$$WER = \frac{S + D + I}{N} \times 100$$

Where:

- S is the number of substitution errors.
- D is the number of deletion errors.
- I is the number of insertion errors.
- N is the total number of words in the ground truth text.

2.3.2 Character Error Rate

Character Error Rate (CER) is similar to WER but operates at the character level instead of the word level. It is particularly useful for evaluating models at a more forgiving level, as sometimes the models can predict lack one character in a word, which makes the whole word regarded as wrong, but our human brain is still able to interpret the word. CER is calculated as:

$$CER = \frac{S + D + I}{N} \times 100$$

Where:

- S is the number of substitution errors.
- D is the number of deletion errors.
- I is the number of insertion errors.
- N is the total number of characters in the ground truth text.

2.3.3 Levenshtein Distance*

The Levenshtein Distance[15], also known as the edit distance, is not an evaluation metric but a measure of the difference between two sequences (typically strings). It is defined as the minimum number of single-character edits required to transform one string into the other. This algorithm was implemented to calculate the "distance" between the predicted text sequence and the ground truth sequence for WER and CER. Given two strings s_1 and s_2 of lengths n and m respectively, the Levenshtein distance lev(s_1, s_2) can be defined recursively as follows:

$$\operatorname{lev}(s_1, s_2) = \begin{cases} \max(n, m) & \text{if } \min(n, m) = 0 \\ \min \begin{cases} \operatorname{lev}(s_1[0 \dots n-1], s_2) + 1 & (Insertion) \\ \operatorname{lev}(s_1, s_2[0 \dots m-1]) + 1 & (Deletion) \\ \operatorname{lev}(s_1[0 \dots n-1], s_2[0 \dots m-1]) + [s_1[n] \neq s_2[m]] & (Substitution) \end{cases} & \text{otherwise} \end{cases}$$

Where:

- s_1 and s_2 are the strings for which you want to compute the Levenshtein distance.
- n and m are the lengths of strings s_1 and s_2, respectively.

Chapter 3

Materials and Methods

In this chapter we will talk about the detailed information on the dataset and the LipNet model, the methods we used to preprocess the data and modify the model. Below is the detailed pipeline for data processing and how the LipNet model works:

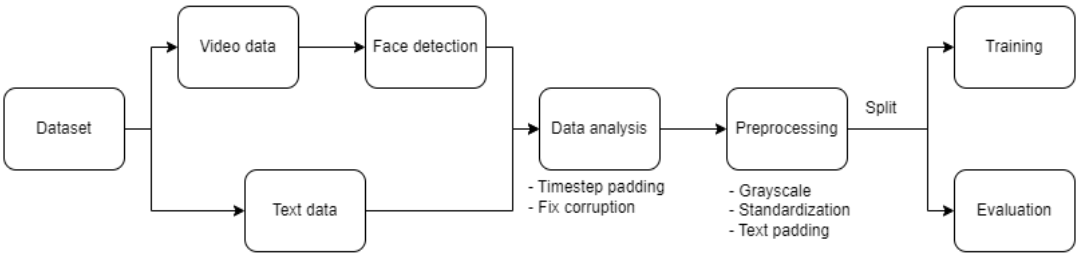


Figure 3. 1 – Overall data processing pipeline

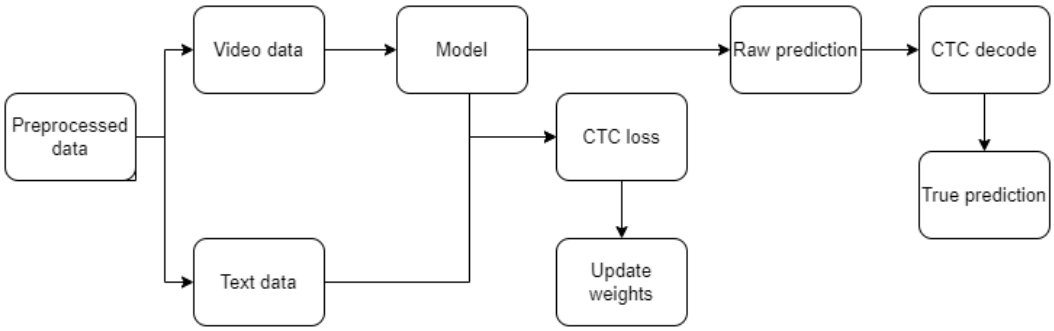


Figure 3. 2 – Model training and predicting process

3.1 Dataset

The Grid Corpus[*] is a large multi-talker audiovisual sentence corpus designed to support joint computational-behavioral studies in speech perception. In brief, the corpus consists of high-quality audio and video (facial) recordings of 1000 sentences spoken by each of 34 talkers (18 male, 16 female), for a total of 34000 sentences. There are 2 directories with their respective extension for each file name the video file with "mpg" extension and the alignments file with "align" extension. The video

[*] Cooke Martin, Barker, Jon, Cunningham Stuart, Shao Xu. The Grid Audio-Visual Speech Corpus. Download link: https://zenodo.org/records/3625687

files each has 360x288 resolution, 25 frame rate and 3 seconds duration which sum up to 75 frames in total. The sentences from the alignments files are drawn from the following simple grammar: command(4) + color(4) + preposition(4) + letter(25) + digit(10) + adverb(4), where the number denotes how many word choices there are for each of the 6 word categories. The categories consist of, respectively, {bin, lay, place, set}, {blue, green, red, white}, {at, by, in, with}, {A, . . . , Z}\{W}, {zero, . . . , nine}, and {again, now, please, soon}. For example, sentences can be form as such: "bin blue by m one soon".

(a)

```
0 23750 sil
23750 29500 bin
29500 34000 blue
34000 35500 at
35500 41000 f
41000 47250 two
47250 53000 now
53000 74500 sil
```

(b)

Figure 3. 3 – Example of a) Video file and b) Alignment file

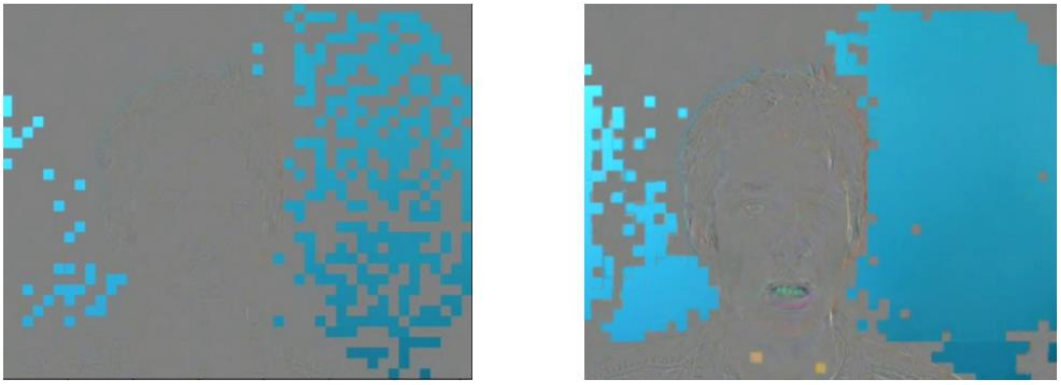
For this internship, we used one male speaker(1) and one female speaker(20) for training and overlapped evaluation, only the test set of one male speaker(2) and one female speaker(22) as unseen data for evaluation, so a total of 4 speakers were used out of 34 speakers. The numbers next to the speakers are their numerical order in the dataset.

3.2 Data Analysis

The original dataset claimed to have 1000 videos for each speaker with each one having exactly 75 frames, but that is not the case for some. There were 5 videos having 74 frames and 3995 videos with 75 frames (namely lrae3s.mpg, sbbbzp.mpg, srbb4n.mpg, srwi5a.mpg, swao7a.mpg). To fix this issue, we decided to duplicate the last frame of each video with 74 frames which should not affect the model's performance since the last few frames are just silence. The goal is to have the input with uniform size with each video having exactly 75 frames.

Aside from the normal ones, there were videos file (namely bbizzn.mpg,

pbio7a.mpg) shown sign of corruption which messed with the face recognition model (more detail on the Data Preprocessing section). By observing the frame with abnormal face bounding box's location and size, we had pinpointed the exact frames that were corrupted in those two video files. The file bbizzn.mpg has corruption ranging from the first frame to the 12th frame, while the file pbio7a.mpg only has the last 3 frames considered as usable.



(a) (b)

Figure 3. 4 – Corruption in video files a) bbizzn.mpg b) pbio7a.mpg

To fix this issue, we set the bounding box recognized at the frame that was not corrupted (in this case we chose the 13th frame) to be fixed throughout the whole video from the first frame to the 75th frame for the bbizzn.mpg file and completely ignore the pbio7a.mpg file as it only has 3 usable frames.

For the alignments, since all 34 speakers utter the same 1000 sentences there was only one folder of 1000 alignment files for the first speaker. To verify the order of video files across all speakers is consistent, we split each speaker’s video files into 10 batches, we watch the first video of each batch and do the same for all speakers. The order is in fact consistent across all speakers, so we load the alignments three more times to fit the amount of video files.

The alignments for this dataset are essentially the ground truth text for the model, we analyzed the number of classes or characters that appear in the original dataset and the alignment with the longest sequence of characters. There are 27 possible classes including all lowercase alphabetic characters from "a" to "z” and the space character " ", the longest sequence observed from the dataset is 31 characters.

3.3 Data Preprocessing

3.3.1 Video Data

Firstly, we need to locate and extract the face region from the data, frame by frame. In this case, we used Google's MediaPipe Face Detection[*] model to locate and draw the bounding box, we will crop each frame to only have the size of the bounding box's width and half of bounding box's height, the half that captures the mouth of the speaker. Since every frame will have different bounding box's width and height, we scanned the face detection through the whole dataset to find the minimum bounding box's width and height detected:

- min_w = 110
- min_h = 55

After getting the min width and min height, we decided to resize each frame to 100 pixels for width and 50 pixels for height. By just resizing the data, we effectively reduce the data size by roughly 4 times (the original resolution is 360x288 while the new resolution is 100x50). The reason we did not resize the cropped region to be larger than the min width and min height is because resizing an image from smaller to bigger (also known as upscaling) can result in a loss of quality and information, so the resized cropped frame must be smaller than the min width and min height.

When it comes to grayscale images, we decided to save two batches, one with grayscale and the other with RGB color channels. We then later train both batches with the same model architecture (only different input layer) and evaluate the results between the two. If gray scaling does not affect the model's performance, it will significantly reduce the training time and data size needed to be load every time. Otherwise, RGB color channels data will be used as creating the best performing model is our utmost priority goal for this internship.

Next, we apply the Z-score Normalization to the whole dataset to have a zero mean and unit variance. This ensures that different features (pixel intensities in frames) are on a similar scale, which helps the model to converge faster and perform better.

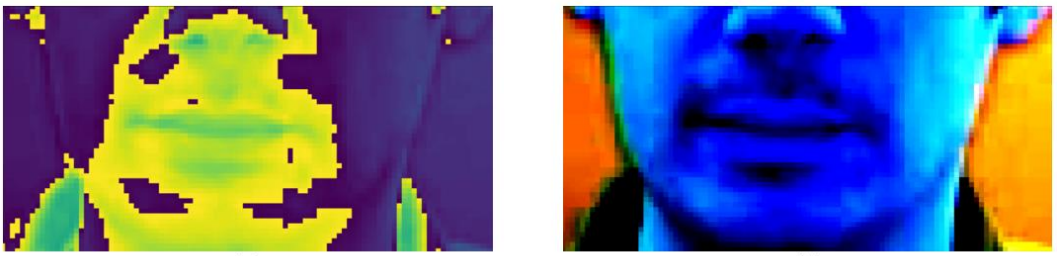


Figure 3. 5 – Video frame after preprocessed a) Grayscale b) GRB

[*] Google AI Developer Team. MediaPipe Face Detection. Guide and API link: https://ai.google.dev/edge/mediapipe/solutions/vision/face_detector

The video data is now ready to train and evaluate on but to further reduce data loading time and computing intensity, we converted the preprocessed data into a Numpy array and save it as a .npy file. Even though, the .npy file will have way more data size than the original dataset because converting video frames to a Numpy array is essentially storing raw pixel data without any compression, but by just loading the .npy file, we will skip the preprocessing part (which took roughly 30 minutes) each time we train and evaluate which is a huge time save.

3.3.2 Alignment Data

As we have the information from the Data Analysis for alignments, there are 27 possible classes or characters and the longest sequence observed contains 31 characters. The number of classes will determine the output of our model which classifies into 29 classes including 26 lowercase alphabetic characters, blank character for padding purpose, space character and a special token for CTC loss function denoted as "-". The reason we need to use blank character for padding is because we need the ground truth data to be of uniform size equals to 31. The categorical data then will be converted into numerical data via label encoding:

Table 3. 1 – Categorical to numerical data conversion

blank “”: 0	“f”: 6	“l”: 12	“r”: 18	“x”: 24
“a”: 1	“g”: 7	“m”: 13	“s”: 19	“y”: 25
“b”: 2	“h”: 8	“n”: 14	“t”: 20	“z”: 26
“c”: 3	“i”: 9	“o”: 15	“u” : 21	space “ “: 27
“d”: 4	“j”: 10	“p”: 16	“v”: 22	CTC token “-”: 28
“e”: 5	“k”: 11	“q”: 17	“w”: 23	

Finally, both the video and the alignment data will be split into train set, validation set and test set. There are 60% of the dataset as train set (1199 videos and alignments), 20% as validation set (400 videos and alignments) and 20% as test set (400 videos and alignments). We only took the train set and validation set from speakers 1 and 20 to train the model, speakers 2 and 22 were held out as unseen data and only use the test set from these two to evaluate the model, which makes our total data size equal to 2799. For comparison between data scale, the original LipNet paper used 28775 videos and alignments only for training excluding data augmentation methods such as flipping the frames, slicing videos to clips of one word and speed tuning video data.

3.4 LipNet Model

LIPNET: END-TO-END SENTENCE-LEVEL LIPREADING paper was authored by Yannis M. Assael, Brendan Shillingford, Shimon Whiteson, and Nando de Freitas[1]. It was published in 2016 and presented at the Advances in Neural Information Processing Systems (NIPS) conference. LipNet is a deep learning model designed for lip reading, which involves recognizing spoken words from the movement of the lips in video sequences.

3.4.1 Model Architecture

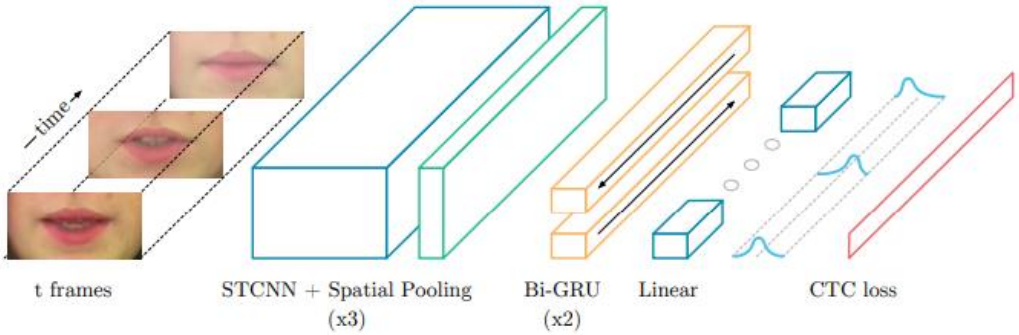


Figure 3. 6 – LipNet model architecture[1]

Figure 3.6 illustrates the LipNet architecture, which starts with 3 layers of STCNN, which include Conv3D filters, ReLu activation functions and spatial max-pooling in each layer. Subsequently, the features extracted are then flattened inside the Time Distributed layer to fit the format of Bi-GRU's input (time-step, sequence of features). The 2 layers of Bi-GRU are crucial for efficient further aggregation of the STCNN output, enhancing the model's understanding of the sequence context. Finally, a linear transformation is applied at each time-step, followed by a softmax function over all the classes originally in the dataset augmented with the CTC special blank token, and then compiled with the CTC loss function. CTC loss is very useful here as it enables sequence-to-sequence learning without requiring frame-level alignment of the input (75 frames which is equivalent to 75 characters after prediction) and the output (text which can only go as long as 31 characters). For example, if the model predicts ["H", "-", "H", "I", "I"], the CTC decoding of this prediction would be "HI" as CTC allows repetition and the special token "-", so as long as the input text size is larger than the output text size, CTC works wonder.

According to the paper, the network parameters of the model were initialized using He initialization[16], apart from the square GRU matrices that were orthogonally initialized, as described in[10]. The models were trained with channel-wise dropout

(dropout rate p = 0.5) after each pooling layer and mini-batches of size 50. They used the optimizer Adam[17] with a learning rate of 10^{-4}, and the default hyperparameters: a first-moment momentum coefficient of 0.9, a second-moment momentum coefficient of 0.999, and the numerical stability parameter epsilon=10^{-8}. This is the detailed hyperparameters table of our approach for the LipNet model:

Table 3. 2 – LipNet detailed model hyperparameters and structure[1]

Layer	Size / Stride / Pad	Input size	Dimension order
STCNN	3 × 5 × 5 / 1, 2, 2 / same	75 × 3 × 50 × 100	T × C × H × W
Pool	1 × 2 × 2 / 1, 2, 2	75 × 32 × 25 × 50	T × C × H × W
Dropout	0.5	75 × 32 × 12 × 25	T × C × H × W
STCNN	3 × 5 × 5 / 1, 1, 1 / same	75 × 32 × 12 × 25	T × C × H × W
Pool	1 × 2 × 2 / 1, 2, 2	75 × 64 × 12 × 25	T × C × H × W
Dropout	0.5	75 × 64 × 6 × 12	T × C × H × W
STCNN	3 × 3 × 3 / 1, 1, 1 / same	75 × 64 × 6 × 12	T × C × H × W
Pool	1 × 2 × 2 / 1, 2, 2	75 × 96 × 6 × 12	T × C × H × W
Dropout	0.5	75 × 96 × 3 × 6	T × C × H × W
Bi-GRU	256	75 × (96 × 3 × 6)	T × (C × H × W)
Dropout	0.5	75 × 512	T × F
Bi-GRU	256	75 × 512	T × F
Dropout	0.5	75 × 512	T × F
Linear	29	75 × 512	T × F
Softmax		75 × 29	T × V

Where T denotes time, C denotes channels, F denotes feature dimension, H and W denote height and width and V denotes the number of words in the vocabulary including the CTC blank symbol. The model has a total of 4574973 parameters.

3.4.2 Training Process

With the complete structure of the LipNet model architecture established, we proceeded to the training session. Each training session took roughly between 4 to 6 hours to train locally and 2 to 3 hours to train on Google Colab, the early stopping save point is set to 10 (if the model does not improve on the validation loss after 10 epochs, the training process will automatically stop and save the best one). Due to the absence of official GitHub source code from the authors of the LipNet paper, our implementation aims to replicate the original model as closely as possible to observe its performance on smaller data scale. The model was trained on RGB color data as

the original structure implied and we also tested whether grayscale data affects the results or not. The model performs worse on grayscale data compared to RGB color data due to converting frames to grayscale leads to the loss of color information, reduction in feature diversity, diminished texture details. Below are the results of our first attempt at replicating the LipNet model using WER and CER for overlapped speakers as evaluation:

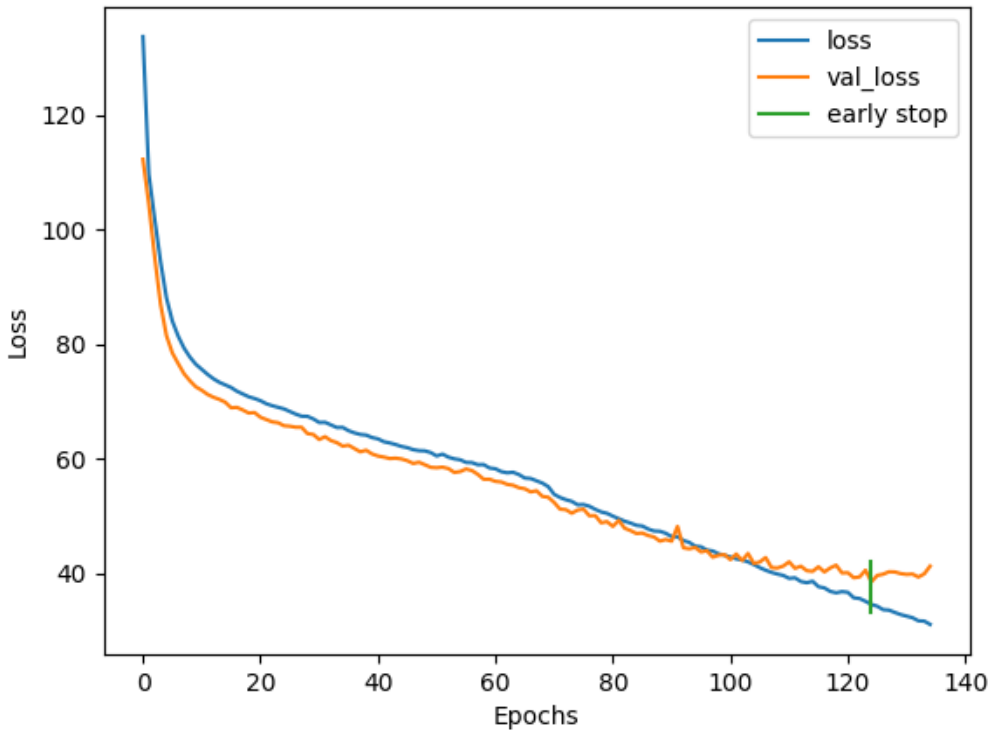


Figure 3. 7 – Learning curve of our attempt at replicating LipNet
Early stopping line at train_loss = 34.51 and val_loss = 38.5

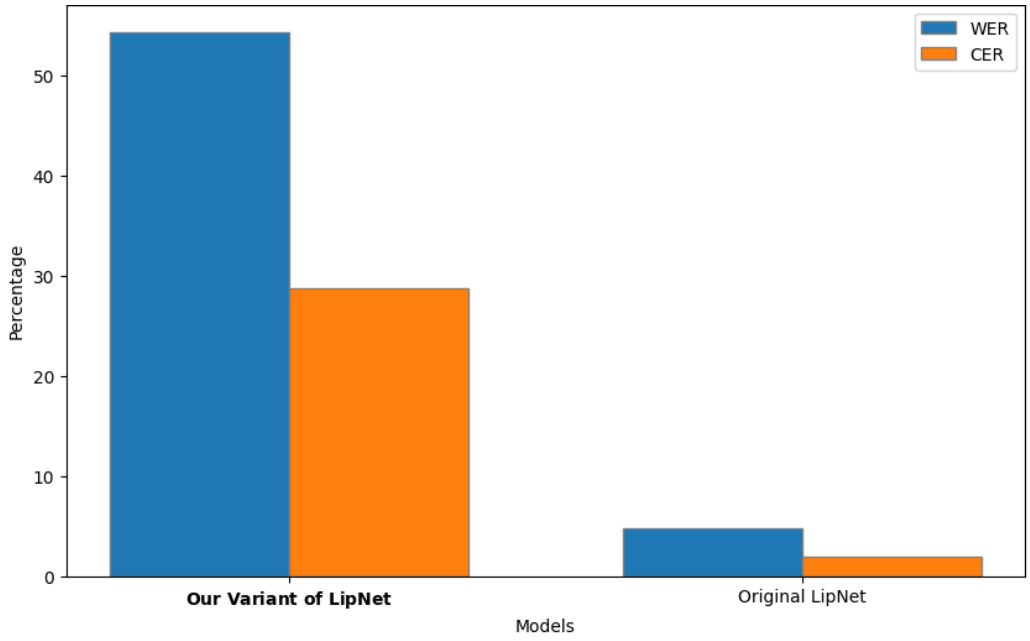


Figure 3. 8 – Comparison of WER and CER between our variant and the original LipNet

At our first attempt, we failed to replicate the model's performance as we expected. There were some factors that contributed to this failure namely: insufficient data size as we trained the model on a much smaller scale compared to the original paper, slightly different standardization hyperparameters in the data preprocessing part, incorrect batch size which can affect the convergence of the model especially on different dataset size, Suboptimal hyperparameters (learning rate, dropout rate, ...) and model structures (different layer types, sizes, or configurations). To improve the results, we decided to modify LipNet by ourselves by finding, learning, and adding new techniques and methods to the model.

3.5 Our Model (Modified LipNet)

This section is when we began to improve the results and spent most of our time on for this internship. Throughout the training process, we monitored the learning curve and evaluated the model's performance after each training session. This iterative approach allowed us to fine-tune the model. Throughout this whole internship, we trained a total of 39 models, we recorded 14 models with improved performance compared to the previous ones and 3 times when the results improved substantially. In this thesis, we will only talk about these 3 times or 3 breakthroughs, how the model architecture changes compared to the previous breakthrough, the learning curve, and the results comparison.

3.5.1 First Breakthrough

This was a big step up from our previous attempt at replicating the LipNet model architecture, we identified noticed some suboptimal hyperparameters related to data size and some issues between the label classes and the CTC loss function. The dataset we used for training differs in size from the one used by the original authors, therefore we adjusted the batch size to prevent the model from updating too frequently, which could lead to overfitting, or too slowly, which could result in prolonged training times and suboptimal learning. The optimal batch size we recorded was 16 out of the set we used (1, 2, 4, 8, 16, 32, 50). As for the issues between the label classes and the CTC loss function, the original paper only uses 28 label classes including 26 alphabetic characters, space " " and CTC blank "-" but our model's output classes is augmented with the blank "" character as padding. The CTC loss function accounts for the blank character as ground truth label to predict which greatly affects the training loss, therefore affecting the performance as in

reality the model does not need to predict the padding section. To fix this problem, we used a mask on the input of the CTC loss function to filter out the blank characters in the ground truth text before calculating the loss. This essentially adjusts the weights of the model so that the probability of predicting a blank character will always be close to or equal to 0 because the ground truth data that go through the loss function do not contain blank characters. Below are the results of our first breakthrough at modifying the LipNet model using only overlapped speakers as evaluation:

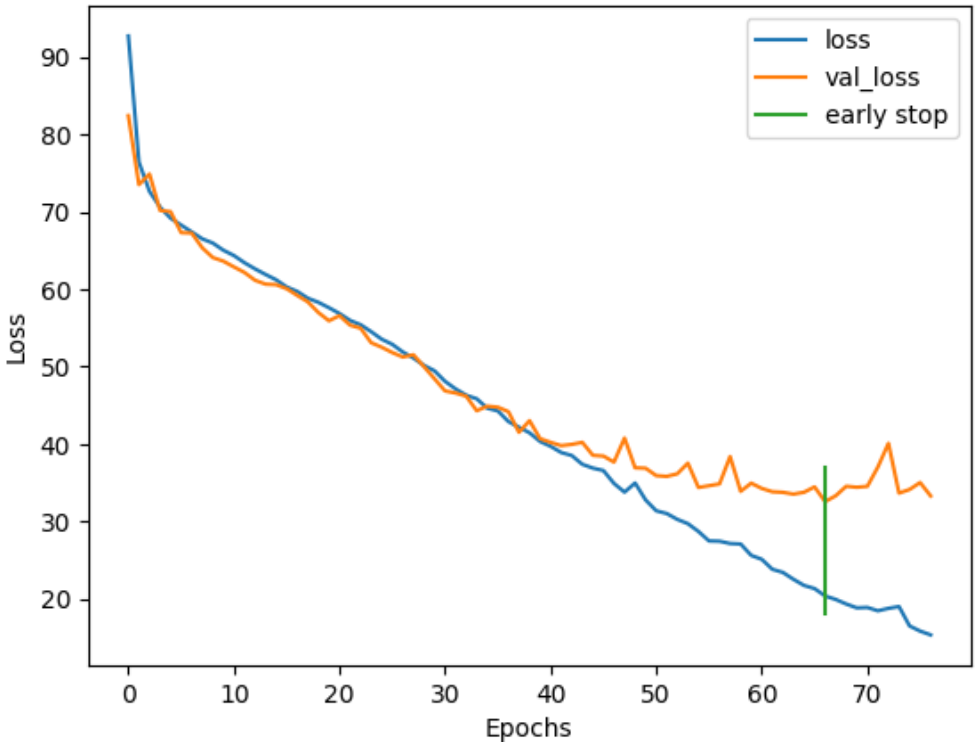


Figure 3. 9 – Learning curve of our first breakthrough modifying LipNet
Early stopping line at train_loss = 20.38 and val_loss = 32.56

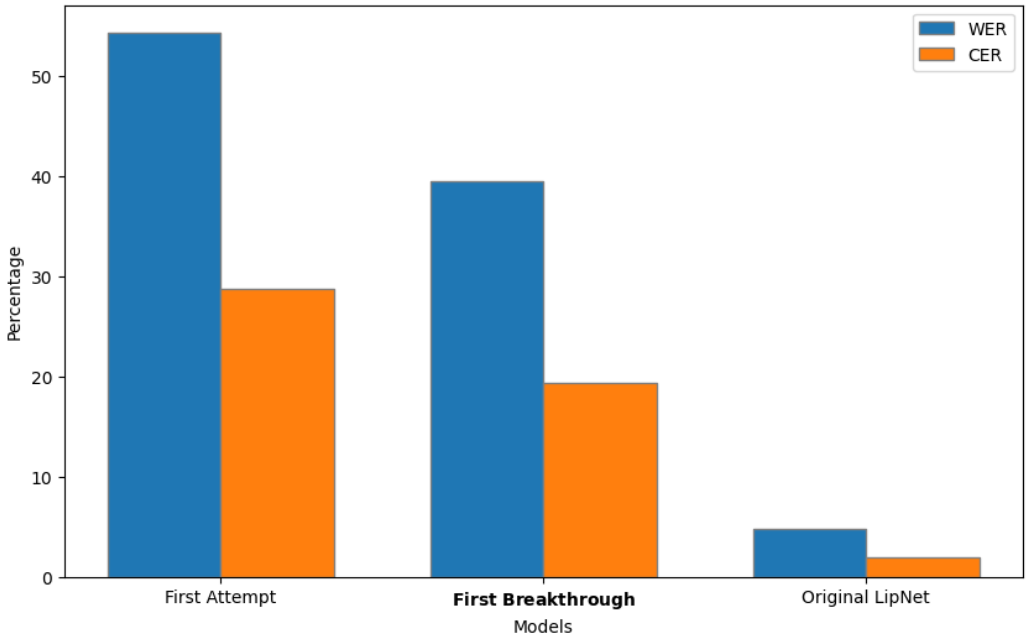


Figure 3. 10 – Comparison of WER and CER between our first breakthrough and previous ones

After the improvement, the training loss and validation loss are overall lower than in the first attempt, but the training loss is still relatively high and validation loss decreases at a slower rate which suggests that the model might be overfitting. The initial WER decreased from 54.4% to 39.5% and CER decreased from 28.8% to 19.4%, therefore the model's performance overall improved by 27.39% and 32.64% respectively.

3.5.2 Second Breakthrough

From the previous results, we observed indications of underperformance and overfitting. Specifically, the training loss remained relatively high, and the validation loss decreased at a slower rate than the training loss. We came to a dead end at that time trying to address the overfitting issue, so we decided to enhance the model's overall performance by decreasing both training loss and validation loss. We removed the Dropout layers between STCNNs and added Batch Normalization layers instead, we also implemented a learning rate scheduler to fine-tune the learning rate dynamically. Initially, Dropout was employed to prevent overfitting, but it was impairing the learning capacity of the model. Therefore, by removing it, the model can fully utilize the STCNNs for more effective feature extraction. Batch Normalization only helps in maintaining a consistent distribution of the data, speeding up training time and model convergence. The learning rate scheduler optimized the learning process, decreasing the learning rate whenever the model hits a plateau for more stable and steady convergence toward the end. We set the hyperparameters so that, every time there is no improvement in the validation loss for 7 consecutive epochs, the learning rate will decrease by a factor of 0.1, until it hits the minimum learning rate of 10^{-6}. Due to the addition of a learning rate scheduler, we adjusted the early stopping point to be 15 epochs instead of 10, for the purpose of updating the learning rate twice before ending the training session when the model hits a plateau. Below are the results of our second breakthrough at modifying the LipNet model using only overlapped speakers as evaluation:

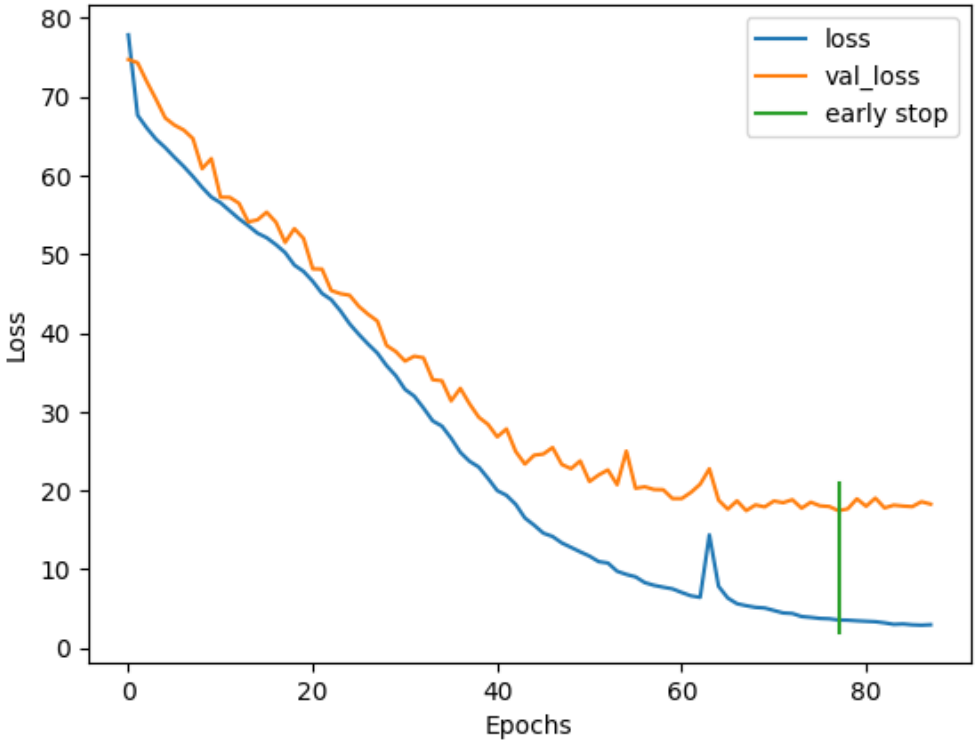


Figure 3. 11 – Learning curve of our second breakthrough modifying LipNet
Early stopping line at train_loss = 3.57 and val_loss = 17.45

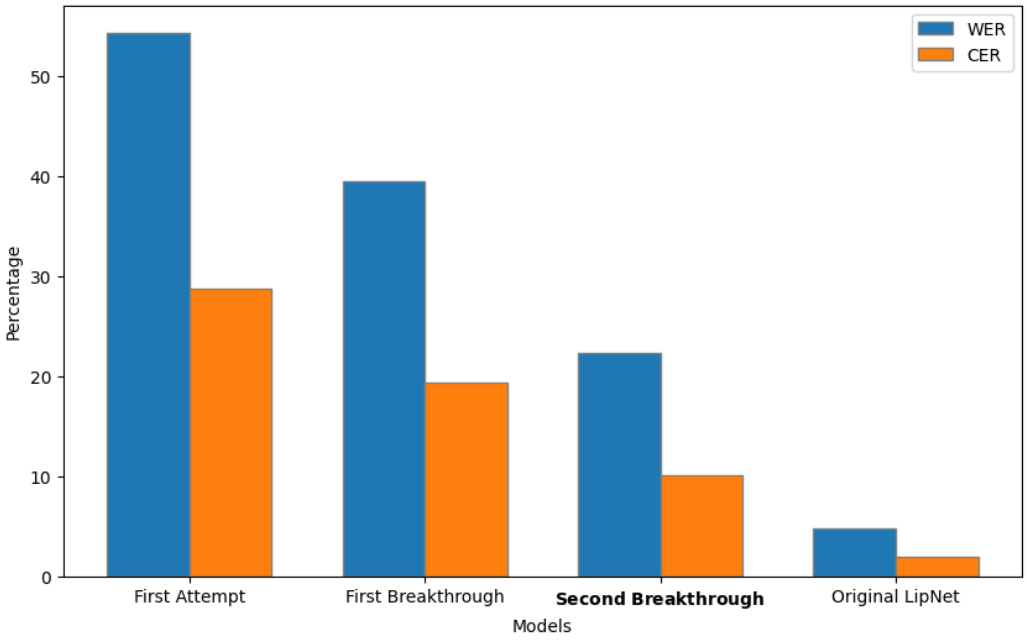


Figure 3. 12 – Comparison of WER and CER between our second breakthrough and previous ones

After the improvement, the training loss and validation loss got significantly lower than in the previous breakthrough. Although, the validation loss still decreases at a slower rate than training loss at one point, which is valid because we have not solved the overfitting issue yet. The WER decreased from 39.5% to 22.4% and CER decreased from 19.4% to 10.1%, therefore the model's performance overall improved by 43.29% and 47.94% respectively.

3.5.3 Third Breakthrough

This is the third and final significant improvement made during this internship, due to the lack of time available. At this point, our model demonstrated excellent performance on the training dataset as the training loss is relatively low, so we decided to tackle the overfitting issue with the goal of minimizing the gap between the training loss and validation loss while maintaining the model's performance on train set. Initially, we employed Dropout layers (p = 0.5) after each max pooling layer for the STCNNs with the hope of replicating the original LipNet model. Upon further research and experimentation, we realized that we (mostly i) misunderstood what the paper was implying, they mentioned that their model was trained using channel-wise Dropout layers or SD3D layers. Unlike standard Dropout, which randomly drops individual neurons, SD3D drops entire feature maps, effectively prevents the model from becoming too reliant on any single feature map, thereby enhancing its ability to generalize. After we were aware of this fact, we tested the implementation of SD3D in our model, either SD3D after every max pooling layer or SD3D after every Conv3D layer in STCNNs. The model with SD3D (p = 0.5) after every max pooling layer performed worse than the results from our first breakthrough and SD3D (p = 0.5) after every Conv3D layer performed better than the results from our second breakthrough by a small margin, so we chose the second option. Even though it was barely better than our model from the second breakthrough, the thing that caught our attention was that, the gap between the training loss and validation loss was basically non-existent, the model converged slower and the performance on train set got worse. It was the right answer to solve the overfitting issue, but the overall performance of the model got worse, therefore we decided to balance out the dropout rate, the optimal dropout rate in our case was 0.2, which means that 20% of the feature maps were dropped during each training iteration. Additionally, we changed the batch size from 16 to 8 to increase the weights update frequency as the model converged rather slowly and inefficiently. Below is the results of our third and final breakthrough at modifying the LipNet model using only overlapped speakers as evaluation:

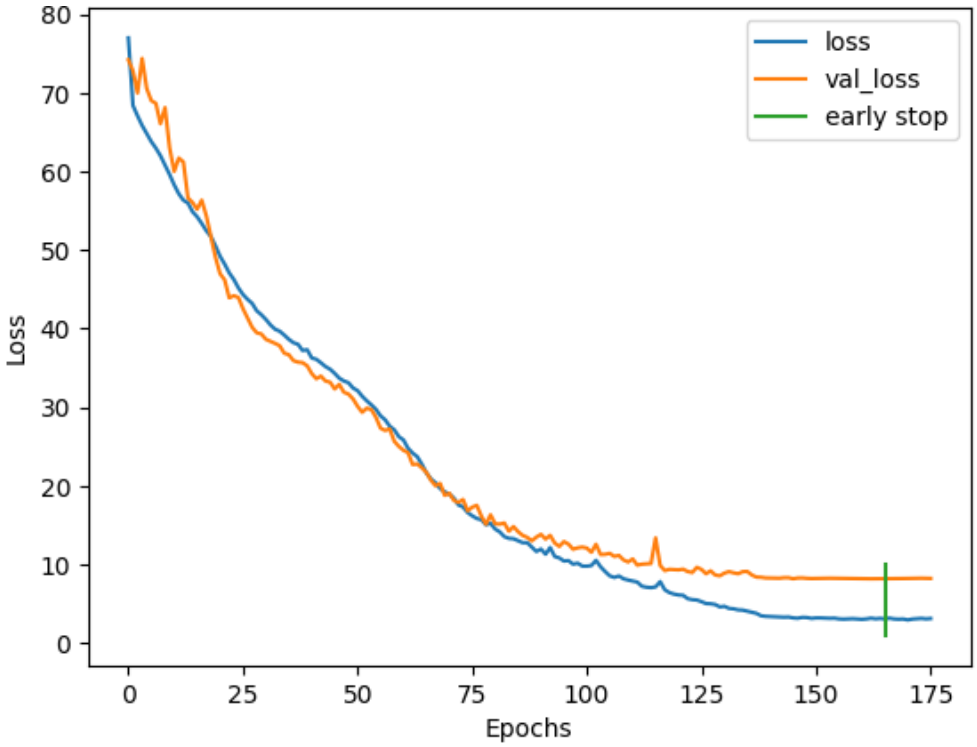


Figure 3. 13 – Learning curve of our third breakthrough modifying LipNet
Early stopping line at train_loss = 3.08 and val_loss = 8.21

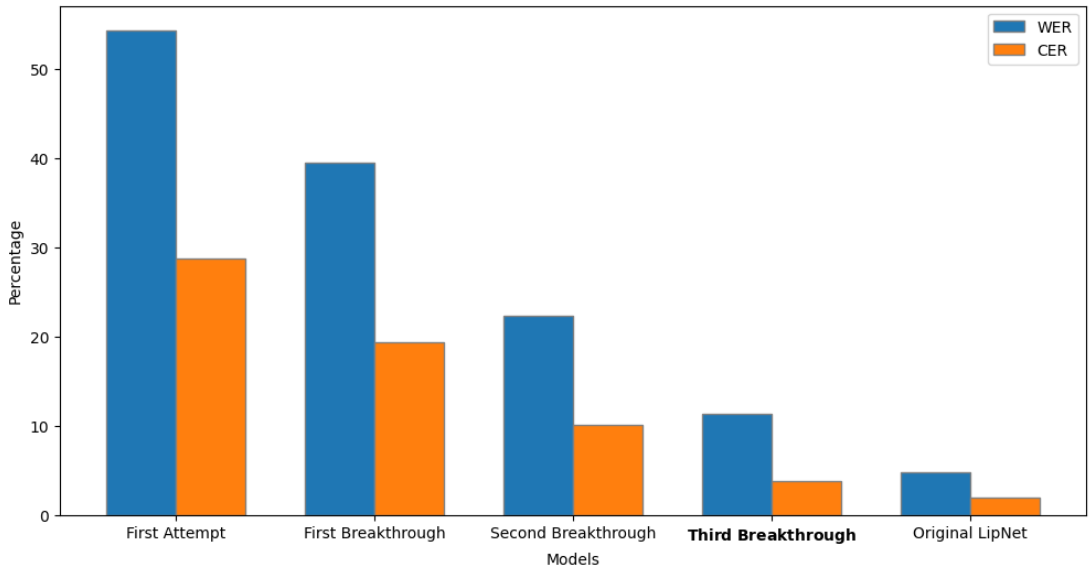


Figure 3. 14 – Comparison of WER and CER between our third breakthrough and previous ones

This is the best model and the best results we could get in our capacity during this internship. The gap between the training loss and validation loss is significantly smaller compared to our previous breakthrough. The WER decreased from 22.4% to 11.3% and CER decreased from 10.9% to 3.8%, therefore the model's performance overall improved by 49.55% and 65.1% respectively.

Table 3. 3 – Detailed hyperparameters and structure of our model

Layer	Size / Stride / Pad	Input size	Dimension order
STCNN	3 × 5 × 5 / 1, 2, 2 / same	75 × 3 × 50 × 100	T × C × H × W
BatchNorm		75 × 32 × 25 × 50	T × C × H × W
SD3D	0.2	75 × 32 × 25 × 50	T × C × H × W
Pool	1 × 2 × 2 / 1, 2, 2	75 × 32 × 25 × 50	T × C × H × W
STCNN	3 × 5 × 5 / 1, 1, 1 / same	75 × 32 × 12 × 25	T × C × H × W
BatchNorm		75 × 32 × 12 × 25	T × C × H × W
SD3D	0.2	75 × 32 × 12 × 25	T × C × H × W
Pool	1 × 2 × 2 / 1, 2, 2	75 × 64 × 12 × 25	T × C × H × W
STCNN	3 × 3 × 3 / 1, 1, 1 / same	75 × 64 × 6 × 12	T × C × H × W
BatchNorm		75 × 32 × 6 × 12	T × C × H × W
SD3D	0.2	75 × 32 × 6 × 12	T × C × H × W
Pool	1 × 2 × 2 / 1, 2, 2	75 × 96 × 6 × 12	T × C × H × W
Bi-GRU	256	75 × (96 × 3 × 6)	T × (C × H × W)
DropOut	0.5	75 × 512	T × F
Bi-GRU	256	75 × 512	T × F
DropOut	0.5	75 × 512	T × F
Linear	29	75 × 512	T × F
Softmax		75 × 29	T × V

Table 3.3 is our finalized model structure and hyperparameters with a total of 4575741 parameters including 384 non-trainable parameters due to Batch Normalization. We used a masked CTC loss function to filter text padding, a learning scheduler (patience = 7, rate = 0.1, min = $10\_^{-6}$), an early stopping callback (patience = 15) and trained the model with batch size = 8.

Chapter 4

Experiment and Results Analysis

4.1 Experimental Setup

All experiments are run either locally on our laptop or on Google Colab. During internship period, training on Google Colab proved to be faster and more useful than training locally by only requiring roughly 1/3 of the time while not consuming our laptop's resources, but Colab has a downside of limiting GPU usage, we can only train max 3 hours and 40 minutes every 2 days. Hence, we switch to train on our laptop whenever we hit Google Colab's GPU limit, while at other times, we would work on other parts of the project while we let Colab do its thing.

To train the model locally, we established a dedicated Conda environment named "lipread" running on Python version 3.10.14. The environment was configured to utilize TensorFlow GPU 2.10.0, with specific versions of cuDNN (8.1) and CUDA (11.2) to ensure compatibility and optimal performance. Incorrectly installation of either cuDNN version or CUDA version will instantly cause errors while using TensorFlow GPU. Our laptop's specifications are as below:

- Operating system: Windows 11 Pro 64-bit
- CPU Processor: Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz (8 CPUs)
- Memory: 16384MB RAM
- GPU Processor: NVIDIA GeForce GTX 1050 Ti
- Display/GPU Memory: 4004MB

Google Colab is a Jupyter Notebook-based platform for data scientists and ML learners with free access to computational resources, including CPUs and limited GPUs. Below are the lists of all the libraries used in our thesis:

- Tensorflow GPU 2.10.0: Deep learning framework.
- OpenCV 4.8.1.78: Read video data.
- Scikit-Learn 1.4.2: Train/validation/test split.
- Numpy 1.24.3: Save and load Numpy arrays.
- Matplotlib 3.8.0: Data and results visualization.

- MediaPipe 0.10.9: Face detection model.
- Editdistance 0.8.1: Auto calculate Levenshtein distance.

4.2 Results Analysis

For evaluating the performance of our lip reading model, we used one male speaker (Speaker 1) and one female speaker (Speaker 20) as overlapped speakers, which means only the train set and validation set were used to train the model and the test set was used as evaluation data. Additionally, to assess the model's generalization capability, we used one male speaker (Speaker 2) and one female speaker (Speaker 22) as unseen speakers, but only on ground truth sentences that the model was not trained on.

We employed two key evaluation metrics to measure the model's performance:

- WER: This metric measures the percentage of words that were incorrectly predicted by the model. It is a standard measure for evaluating the performance of speech and lip reading systems.

- CER: This metric measures the percentage of characters that were incorrectly predicted by the model. It provides a finer-grained analysis of the model's performance, capturing errors at the character level rather than at the word level.

4.2.1 Comparison With Different Studies

To evaluate LipNet, we compare its performance to that of three hearing-impaired people who can lipread, the original LipNet paper and two other models that use the same evaluation metrics and dataset as our project. The methods and statistics on how hearing-impaired people and two other models did it were originated from the LipNet paper.

Hearing-Impaired People[1]**:** After being introduced to the grammar of the GRID corpus, three members of the Oxford Students' Disability Community observed 10 minutes of annotated videos from the training dataset, then annotated 300 random videos from the evaluation dataset. When uncertain, they were asked to pick the most probable answer.

Lipreading With Long Short-Term Memory (Wand et al)[18]**:** In the LipNet paper, they replicated the model architecture from the original paper and trained it the same way as LipNet. The model uses two LSTM layers with 128 neurons. The input frames were converted to grayscale and were down-sampled to 50×25px, dropout p = 0, and the parameters were initialized uniformly with values between [−0.05, 0.05].

Lip Reading Sentences in the Wild (Chung et al)[19]**:** In the LipNet paper, they replaced the STCNN with spatial-only convolutions similar to those of the model architecture from the original paper and trained it the same way as LipNet.

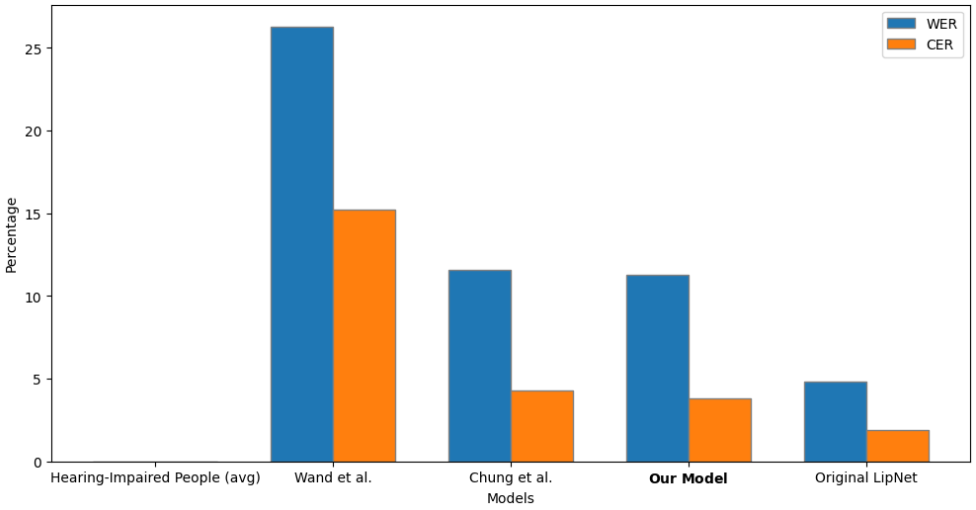


Figure 4. 1 – Comparison of WER and CER on overlapped speakers

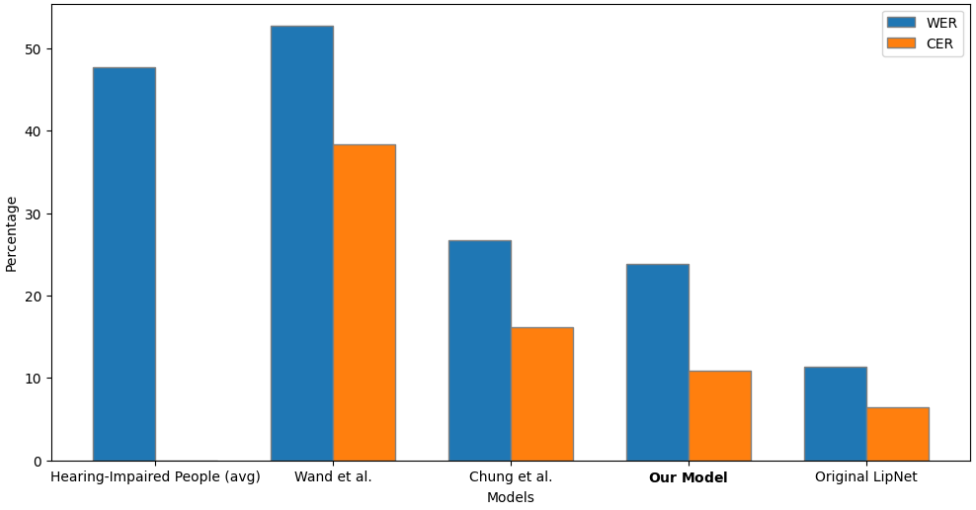


Figure 4. 2 – Comparison of WER and CER on unseen speakers

Table 4. 1 – Detailed comparison of WER and CER with different studies

Method	Overlapped speakers		Unseen speakers	
	WER	CER	WER	CER
Hearing-Impaired People (average)	None	None	47.7	None
Wand et al.	26.3%	15.2%	52.8%	38.4%
Chung et al.	11.6%	4.3%	26.7%	16.2%
Our model	11.3%	3.8%	23.8%	10.9%
LipNet	4.8%	1.9%	11.4%	6.4%

The hearing-impaired group does not have a designated unseen dataset or CER calculation because the evaluation of hearing-impaired individuals focuses on their real-time lip reading ability without distinguishing between seen and unseen speakers, and CER is unnecessary here because if they can predict the word correctly, they can form the characters to make that word with ease.

4.2.2 Discussion

In this section, we will discuss about the results from figure 4.1, 4.2 and table 4.1:

- The WER and CER of all models on unseen data are higher than overlapped data for obvious reasons, the models were not trained on the speakers from unseen data, but this scenario tests the model's generalization ability, which is crucial for real-world applications.
- The WER for hearing-impaired people (average) is the highest, highlighting the challenge in lip reading in general for human beings.
- Despite training on a smaller data scale than other models, our model managed to achieve a lower WER and CER compared to Wand et al. and Chung et al., only get beaten by the original LipNet's performance.
- Overall, our model performed decently on both overlapped and unseen speaker datasets, a moderately successful experiment for us.

Chapter 5

Deployment

In this chapter, we will talk about the process of deploying our model to a desktop application. Overall, the pipeline below shows how our application works in general, also the parts we have not worked on or finished yet in this internship:

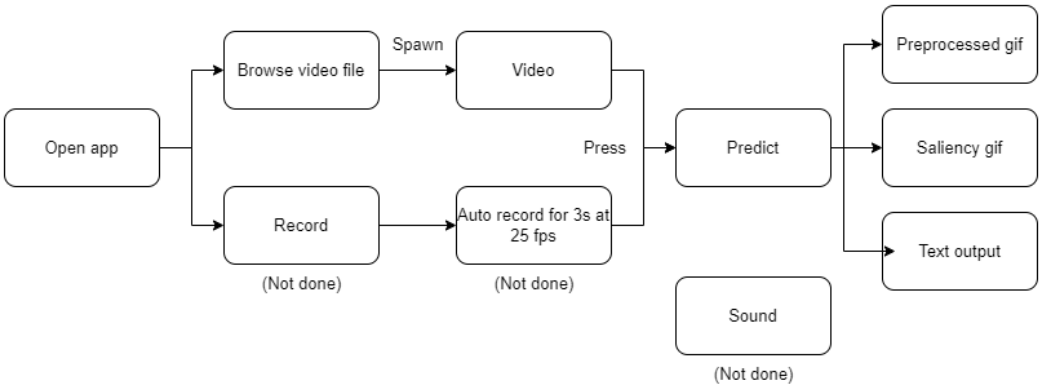


Figure 5. 1 – Application pipeline

5.1 Front-End

We built the front-end of this application using Streamlit, a powerful and easy-to-use framework for creating web applications in Python. The layout is designed as below:

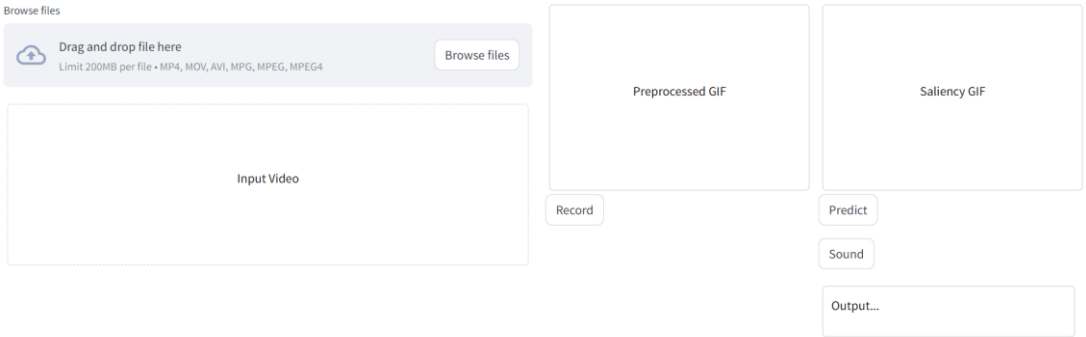


Figure 5. 2 – Based UI upon opening the application

We set a wide layout and split the tab into 3 columns or sections:

- The first column: a drag and drop file function allowing MP4, MOV, AVI, MPG, MPEG, MPEG4 extension files with a browse button to browse video files, it will show the file name upon select a file; the box with "Input Video" as

text is a placeholder for when done browsing, it will replace the box with the actual chosen video to allow playback.

- The second column: the box named "Preprocessed GIF" as placeholder for the preprocessed output of the chosen video; the "Record" button intended to auto record for 3 seconds upon pressed and save the video locally, but we have not finished the function for that button yet.
- The third column: the box named "Saliency GIF" as placeholder for the saliency map between the gradient after predicting and the original input, formatted as a gif file; the "Predict" button upon pressed, will spawn the 2 gif files above and the text prediction; the "Sound" button supposedly covert the text prediction into sound via Text-To-Speech, which we also have not finished yet; The "Output" box is used to store the final text output of the model.

Overall, after pressing the "Predict" button our application will look something like this:

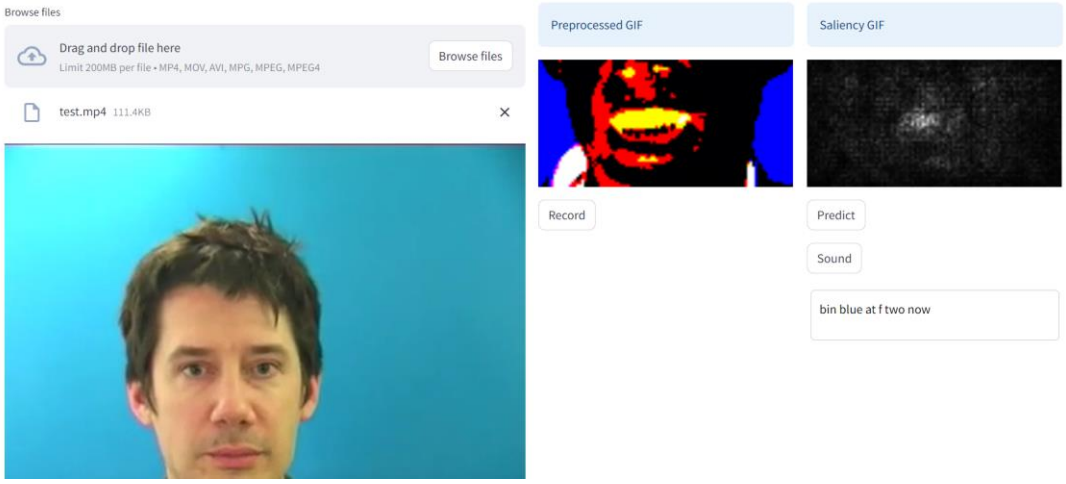


Figure 5. 3 – How the application looks after predicting

5.2 Back-End

The back-end of the application is responsible for processing the chosen video file such as preprocessing the input video and creating saliency map upon pressing "Predict" button, generating text predictions, and converting text to sound (not finished).

Video processing:

- **Preprocessed GIF:** After browsed the desired file, the system stores the path and pushes the path to the preprocess function as input, the function then outputs a set of Numpy array shape (75, 50, 100, 3) and the output is converted and saved locally as a gif file using the imageio library. The system will spot the gif name and replace the "Preprocessed GIF" to the actual gif file.
- **Saliency GIF:** Basically, the same process as preprocess but, to create a saliency map, we need both the input video and the output prediction to calculate loss and gradient. Initially, we would put the saliency map overlay with the input video, but we have not succeeded on that task yet, so we just saved the whole saliency map shape (75, 50, 100) as a gif to replace the "Saliency GIF" box upon pressing the predict button.

Generating Predictions: The process of generating predictions involves getting the output from the preprocess function then load the model to predict on the preprocessed data, we then save the prediction to a .txt file so that the system can read from that and put the text inside onto the "Output" box.

We were supposed to have a functioning “Record” button and “Sound” button to increase user-friendliness and effectiveness, but we did not have the time to complete their function. The “Record” button supposed to automatically pop up a tab that shows the recording of the user then automatically close after 3 seconds, the system will take that recording as input instead of having the user to record by web camera then browse the file later. The “Sound” button supposed to trigger the text output to sound conversion function utilizing Text-to-Speech API such as OpenAI to output robust sound.

5.3 Standalone Executable Conversion

After we made sure that our application run smoothly on local Streamlit server, we deploy the web application as a standalone executable (.exe) file using PyInstaller library. It is a very easy process as we just need to put this command line "pyinstaller --onefile app.py" into the terminal with “app.py” being our application's filename, then we let Pyinstaller do all the necessary jobs to build a .exe file. It will open the web application similar to when running the Streamlit script in terminal but with just a single click to the app.exe file instead.

Chapter 6

Conclusion and Future Works

6.1 Summary of Thesis

We successfully modified the LipNet model to perform on smaller data scale with decent results, surpassing other studies except LipNet on the same dataset with less data used for training. We have learned about various model optimizing and hyperparameter tuning techniques throughout this internship and I as a student have learned to build, test, and evaluate my first DL model in Python. We also built a working lip reading translation desktop application using our own model, it is not fully done yet but we are at least 80% there to complete the application. We hope upon finishing, it will help the hearing-impaired people to communicate better and serve as an alternative way to communicate in noisy environments.

6.2 Future Works

Many developments, tests and experiments have been left for the future due to lack of time. These are the ideas that we may develop and experiment in the future:

- Finishing the application is our utmost priority, that includes the "Record" button function, the "Sound" button function and saliency map overlay.
- Increase the model's performance by implementing Attention mechanism[20] to our model's structure.
- Try to experiment replacing GRUs to Transformers structure[21].
- Deploy the application on Mobile and other platforms.

Bibliography

[1] Yannis M. Assael, Brendan Shillingford, Shimon Whiteson, Nando de Freitas. “LIPNET: END-TO-END SENTENCE-LEVEL LIPREADING”. In GPU Technology Conference (GTC), pp. 1599-1611 (2017).

[2] Hangchong Sheng, Gangyao Kuang, Liang Bai, Chenping Hou, Yulan Guo, Xin Xu, Matti Pietikäinen, and Li Liu. “Deep Learning for Visual Speech Analysis: A Survey”. In Pattern Recognition (PR), pp. 6-10 (2022)

[3] Liang Dong, Say Wei Foo, Yong Lian. “A Two-Channel Training Algorithm for Hidden Markov Model and Its Application to Lip Reading”. In IEEE Transactions on Signal Processing, pp. 566-571 (2005).

[4] Ayaz A. Shaikh, Dinesh K. Kumar, Wai C. Yau; M. Z. Che Azemin, Jayavardhana Gubbi. “Lip Reading Using Optical Flow and Support Vector Machines”. In 2010 3rd International Congress on Image and Signal Processing, (CISP), pp. 301-305 (2010).

[5] Son N. Tran. “Linear-Time Sequence Classification using Restricted Boltzmann Machines”. In IEEE Transactions on Neural Networks and Learning Systems, pp. 3625-3630 (2018).

[6] Alexandros Potamianos, Member, IEEE, and Shrikanth Narayanan, Senior Member, IEEE. “Robust Recognition of Children’s Speech”. In IEEE Transactions on Speech and Audio Processing, pp. 603-615 (2003).

[7] Triantafyllos Afouras, Joon Son Chung, Andrew Zisserman. “Deep Lip Reading: A Comparison of Models and An Online Application”. In 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3493-3501 (2018).

[8] Saravanan Chandran. “Color Image to Grayscale Image Conversion”. In IEEE Transactions on Image Processing, pp. 2482-2490 (2017).

[9] S.Gopal Krishna Patro, Kishore Kumar Sahu. “Normalization: A Preprocessing Stage”. In IEEE Transactions on Neural Networks, pp. 1235-1240 (2015).

[10] Junyoung Chung Caglar Gulcehre KyungHyun Cho, Yoshua Bengio. “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling”. In NIPS 2014 Workshop on Deep Learning and Representation Learning, pp. 1-9 (2014).

[11] Alex Graves, Santiago Fernandez, Faustino Gomez, Jurgen Schmidhuber. “Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks”. In International Conference on Machine Learning (ICML), pp. 369-376 (2006).

[12] Sergey Ioffe, Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In International Conference on Machine Learning (ICML), pp. 448-456 (2015).

[13] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In Journal of Machine

Learning Research, pp. 1929-1958 (2014).

[14] Karen Simonyan, Andrea Vedaldi, Andrew Zisserman. “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”. In International Conference on Learning Representations (ICLR), pp. 1-8 (2014).

[15] Shengnan Zhang, Yan Hu, Guangrong Bian. “Research on string similarity algorithm based on Levenshtein Distance”. In International Journal of Computational Theory and Engineering, pp. 161-165 (2015).

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In IEEE International Conference on Computer Vision (ICCV), pp. 1026-1034 (2015).

[17] Diederik P. Kingma, Jimmy Lei Ba. “ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION”. In International Conference on Learning Representations (ICLR), pp. 1-15 (2015).

[18] Michael Wand, Jan Koutnık, Jurgen Schmidhuber. “Lipreading With Long Short-Term Memory”. In IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 6115-6119 (2016).

[19] Joon Son Chung, Andrew Senior, Oriol Vinyals, Andrew Zisserman. “Lip Reading Sentences in the Wild”. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3444-3453 (2017).

[20] Xiang Ye, Zihang He, Heng Wang, Yong Li. “Towards Understanding the Effectiveness of Attention Mechanism”. In IEEE Transactions on Neural Networks and Learning Systems, pp. 388-399 (2018).

[21] Pablo-Andrés Buestán-Andrade, Matilde Santos Peñas, Jesús-Enrique Sierra-García, Juan-Pablo Pazmiño-Piedra. “Comparison of LSTM, GRU and Transformer Neural Network Architecture for Prediction of Wind Turbine Variables”. In IEEE Access, pp. 178945-178955 (2021).