END-TO-END AUTOMATED SPEECH RECOGNITION USING A CHARACTER BASED SMALL SCALE TRANSFORMER ARCHITECTURE

by

Alexander Loubser

Submitted in partial fulfillment of the requirements for the degree
Master of Engineering (Electronic Engineering)

in the

Department of Electrical, Electronic and Computer Engineering
Faculty of Engineering, Built Environment and Information Technology

UNIVERSITY OF PRETORIA

February 2024

SUMMARY

End-to-end automated speech recognition using a character based small scale transformer architecture

by

Alexander Loubser

Supervisor:	De Villiers, Pieter
Co-supervisor:	Dr. A. de Freitas
Department:	Electrical, Electronic and Computer Engineering
University:	University of Pretoria
Degree:	Master of Engineering (Electronic Engineering)
Keywords:	Speech recognition, transformer, end-to-end, character based, connectionist temporal classification, small-scale, LibriSpeech, convolutional neural network

The objective of the research is to determine if it is possible to build a small-scale speech recognition system that is comparable in performance and word error rate, to larger modern automated speech recognition (ASR) systems. The hypothesis is that a small transformer-based ASR model will produce comparable results, in word error rates, to traditional ASR models and compete with modern ASR systems that are exponentially larger in computational size. A small-scale, character-based transformer architecture is developed to create an end-to-end automated speech recognition model. The model architecture consists of a small convolutional neural network and transformer architecture, consisting of 2.214 million parameters, compared to modern speech recognition systems that have over 300 million parameters.

The resulting end-to-end transformer based ASR model designed in this research consists of a convolutional neural network layer and a two-headed transformer with three encoder layers and three decoder layers. The data used to train the model consisted of 2000 hours of Mozilla common voice 7.0 training data and the 1000 hours of LibriSpeech English training data. Audio data is received as input data

to the model. The output of the model has 30 character-based classes. The output of the model is processed by a small 4-gram language model to improve word understanding as it is a character-based ASR model.

The model is trained on limited audio of about 3000 hours, where modern systems are trained with over 50 000 hours of speech data. The common voice and LibriSpeech training datasets were used for training and produced a character error rate and word error rate of 27.89% and 54.5% respectively on the common voice testing data as well as a word error rate of 16.03% and 35.51% on the LibriSpeech test-clean and test-other datasets. An existing gated recurrent unit architecture produced a word error rate of 11.9% and 31.1% respectively on the same LibriSpeech dataset, but at a significantly larger computational cost, as the architecture had 52.5 million parameters, which is a factor of 24 times larger than the architecture in this research. Modern transformer architectures produced a word error rate of between 2% and 4% on the LibriSpeech test-clean dataset, but these architectures are 200 times larger and are trained on 53 000 hours more data than the architecture from this research.

The main application of a small-scale ASR model is for use cases where practitioners do not have access to industrial scale datasets and computing resources. The ASR model is a proof-of-concept for under resourced languages that do not have large corpuses available for training large-scale speech recognition and language models.

LIST OF ABBREVIATIONS

ASR	Automated speech recognition
BERT	Bidirectional encoder representations from transformers
MMI	Maximum mutual information
CER	Character error rate
CNN	Convolution neural network
CTC	Connectionist temporal classification
DCT	Discrete cosine transform
DNN	Deep neural network
FFNN	Feedforward neural network
FFT	Fast Fourier transform
FIR	Finite impulse response
GELU	Gaussian error linear unit
GLUE	General language understanding evaluation
GMM	Gaussian mixture model
GPT	Generative pre-trained transformer
GPU	Graphics processing unit
GRU	Gated recurrent unit
HMM	Hidden Markov model
LPC	Linear predictive coefficient
LSTM	Long-short term memory
MFCC	Mel frequency cepstral coefficient
NLP	Natural language processing
OOV	Out of vocabulary
PLP	Perceptual linear prediction
RNN	Recurrent neural network
STFT	Short time Fourier transform
VRAM	Video random access memory
WER	Word error rate

LIST OF SYMBOLS

α_{filt}	Filter constant
α_{lang}	Weighting ratio for language model probability
α_{mod}	Modulo time step
α_{mom}	Momentum hyperparameter
β	Learnable hyperparameter
β_{lang}	Weighting ratio for language model words in a sequence
δ	Error signal
ε	Extra constant value
γ	Learnable hyperparameter
μ	Mean
ν	Learning rate
Φ	Gaussian cumulative distribution function
σ	Softmax function
a_{ij}	Attention weight
$\mathbf{b}$	bias vector
C	Channels
C_n	Correct Words
Ct	MFCC coefficients
d_k	Dimension of key vector
D_n	Deletions
Dt	Detla MFCC
E_{sse}	Sum of square error cost function
ED	Edit Distance
$E[x]$	Mean of the probability distribution
f_{conv}	Convolving filter size
F_{max}	Maximum frequency
f_{new}	New frequency
f_{orig}	Original frequency
F_{samp}	Sampling frequency
$h_{ctc}(x)$	CTC classifier function

$H_m(k)$	Filterbank
$h(n)$	Filtering function
i	Instance
I_n	Insertions
k	Discrete frequency index
$\mathbf{k_i}$	Key vector
L	Input sequence length
ls	Loss
L'	Layer
l_{in}	Length dimension of input
l_{out}	Length dimension of output
l_{pad}	Padding size
l_{stride}	Stride length
L_{win}	Window length
$M(f)$	Mel scale
$M_{spec}(n)$	Mel spectrogram
Mag	Magnitude
$move_{avg}$	Moving average
n	Sample point
N_{bat}	Batch size
N_{class}	Number of output classes
N_{del}	Number of delta features
N_{Mel}	Number of Mel features
N_{MFCC}	Number of MFCC features
N_{samp}	Sample length
N_w	Number of words in reference sequence
p	Probability
P	Phase
$P(W)$	Prior probability
$P_i(k)$	Power Spectral Estimate
PE	Positional encoding
$PP(W)$	Perplexity
$\mathbf{q_i}$	Query vector

R_{XY}	Cross-correlation function
S_{dat}	Dataset
S_{dft}	Complex discrete Fourier transform
S_n	Substitutions
t	Time based input
T_s	Time step
V	Vocabulary
$\mathbf{v_i}$	Value vector
$\mathrm{Var}[x]$	Variance of the probability distribution
w	Weight
W	Word
$\mathbf{W}$	Weight matrix
w_{in}	Width dimension of input
$\mathbf{W_k}$	Key weight matrix
w_{out}	Width dimension of output
$\mathbf{W_q}$	Query weight matrix
$\mathbf{W_v}$	Value weight matrix
wd	Weight decay
$\mathbf{x}$	Input vector
X_{DCT}	Discrete cosine transform
$x(i)$	Input signal
$x_{sinc}(t)$	Sinc-interpolated input signal
$\mathbf{y}$	Output vector
$y(i)$	Output signal
$\mathbf{z}$	Target vector
z_i	Zero

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

1.1 PROBLEM STATEMENT

1.1.1 Context of the problem

Automated speech recognition (ASR) is the process of translating a spoken language or command into text using computer processing techniques [1]. End-to-end automated speech recognition uses modern architectures, such as transformers, to directly translate the audio data into text without the need of phoneme or lexicon data.

Speech recognition is used for many applications. One popular application is to generate text that can be displayed along with audio for closed captions and subtitles. Other modern applications are voice commands for smart devices and automatic translation. There are multiple reasons why subtitles are added to multimedia and videos. The main reason is to be able to understand and interpret the video that is being displayed. In [2], it is predicted that about 15% of Americans have hearing impairments. Therefore, it is necessary to make videos understandable for the hearing impaired. Subtitles and closed captions can make it possible for anyone with a hearing disability to still enjoy video content and understand the context and content of the video [3]. This creates more viewers and therefore more revenue. Some countries and multimedia services require all media that is broadcasted to include subtitles by law. Text can be generated from audio files using automated audio processing and machine learning. This method reduces the time it takes to create text files as the files are automatically generated. The automation process also reduces the human input required. Any audio file containing spoken language can theoretically be used and text can automatically be generated for the audio files based on the available data [4].

Traditionally, ASR models consisted of separate acoustic and language models. Large acoustic models such as [5] and [6], consist of very large deep neural networks (DNNs) that use large hidden Markov

models (HMMs) and feedforward neural layers to represent the mapping between phonemes and the audio signals or features. The phonemes are converted into words using large language model dictionaries to produce an ASR system. The traditional ASR systems require an exceptionally large model with multiple layers and billions of parameters to produce satisfactory results. Recurrent neural network (RNN) models and long short-term memory (LSTM) models made a significant impact on ASR, as the models could be trained to take the context of the speech into account by training the model with sequential speech data [7, 8, 9]. A language model compares the probability of dependence between the predicted words in a sentence. The model uses the previous words in the sentence to predict the correct form of each following word in a sentence. Language models also improved significantly with the introduction of RNNs, as the method of predicting words was based on longer contextual sequencing around the word, when compared to original n-gram language models. Language models are used to predict the correct text after the acoustic model has predicted the phonemes from the extracted audio features. In [10] and [4], the authors compare n-gram language models to RNN language models. The n-gram models perform better on small-scale language models, but have diminishing gains based on data sizes. RNN language models perform better with larger datasets but require more computational power and training time. In [11] and [12], LSTM language models are shown to perform better than n-gram language models like [13], and RNN language models.

The introduction of attention-based models such as transformers from [14], allowed ASR models to be more efficient by delivering the same results with less computational power required. The transformer architecture was first developed for machine translation and text based tasks where it improved the results compared to any previous RNN and CNN networks [15]. Transformer based models use encoders and decoders linked with a self-attention system and improve on RNN and CNN models due to the recurrence and convolution steps falling away. This allows more sequential data to be used with less memory or parameters required. Modern end-to-end speech recognition systems cut out the phoneme step to reduce the complexity of the system and to reduce error propagation. Attention based architectures make this possible without the use of a very large RNN models. In this research, a small character-based ASR model is designed and implemented, using a transformer architecture, and is compared to larger transformer-based models and RNN models. The model consists of a small convolutional neural network and a transformer network to predict characters based on the input audio features. The input audio features and text are aligned using connectionist temporal classification (CTC) from [16].

ASR requires large datasets and different machine learning algorithms to successfully translate audio into text. Feature extraction takes place by separating the audio data, based on audio properties such as pitch and frequency. Standard audio features are extracted from the raw audio data such as the Mel frequency cepstral coefficient (MFCC) and the delta coefficients the MFCC values [17]. The audio features are selected to reduce the amount of input features while still preserving most of the speech signal. Multiple performance metrics are considered for evaluating the ASR models to ensure the model evaluation is accurate. The word error rate (WER) and character error rate (CER) are performance metrics to determine the percentage of words and characters that were predicted correctly.

1.1.2 Research gap

Text is created for audio and video files to interpret the audio without sound. This is an added function in the media industry to ensure that people from different language backgrounds or people with hearing impairments can understand the video and audio content. There are multiple reasons why text is added to multimedia and videos. The main reason is to be able to understand and interpret the audio that is being presented. Text can be generated using automated audio processing and machine learning. This method reduces the time it takes to create text or subtitle files as the files are automatically generated. The automation process also reduces the human input required. The goal of the research is to analyse audio tracks and perform natural language processing techniques on the audio tracks.

Traditional automated speech recognition models are not very accurate in terms of word error rates, therefore there is an opportunity for new development in the research field of automated speech recognition [7]. There are different approaches to creating an automated speech recognition model. Most modern automated speech recognition models use only a single model to predict text from speech, as the model is required to be simple with the amount of training required [5]. The research implemented will focus on designing state-of-the-art end-to-end model, using a transformer architecture, to create an accurate speech recognition model. This is possible due to more processing power being available of computations than in previous years as well as the simplification of language model architectures.

Modern ASR models are very large and contain more than 100 million parameters in each model. These models are almost impossible to train without industrial scale computer resources and very large corpuses of training data. Modern ASR models use thousands of hours of training data, which is not accessible when working with under resourced languages. The objective of this study is to develop a

small-scale ASR model for use cases where practitioners do not have access to industrial scale datasets and computer resources. The ASR model is a proof-of-concept for under resourced languages that do not have large corpuses available.

1.2 RESEARCH OBJECTIVE AND QUESTIONS

The main objective of the research is to determine if it is possible to build a small-scale speech recognition system that is comparable in performance and word error rate (WER) to larger modern ASR systems. The hypothesis is that a small transformer-based ASR model will produce similar results, in word error rates, to traditional ASR models. The small-scale transformer-based ASR model will compete with modern ASR systems that are exponentially larger in computational size, by combining an acoustic model and a language model for an end-to-end automated speech recognition system. This should improve the word error rate performance of the automated speech recognition system, when compared to available automated speech recognition systems. Different architectures will be investigated to determine if a transformer-based ASR model can outperform other existing architecture models. The study should also determine if a small-scale ASR system is a viable approach for under resourced languages with only small datasets available. Lastly the impact of the amount of training data will be investigated to determine if small datasets can produce relatively accurate CERs and WERs, when compared to larger datasets. During the course of the study, the following research questions will be addressed:

- Is it possible and viable to reasonably combine an acoustic model and a language model into one speech recognition model that will successfully convert speech to text?
- What will the effect of combining an acoustic and language model be to the word error rate of an automated speech recognition system?
- How does a transformer model compare to other existing models when applied to speech recognition processing?
- What is the most suitable neural network algorithm for acoustic modelling?
- Which audio feature extraction methods are the most efficient for natural language processing?
- How accurate does an automated speech recognition model have to be, to increase efficiency and reduce human input required?
- Will a small transformer-based model produce a word error rate performance, that is comparable and similar to larger modern automated speech recognition systems?
- Is it possible to build an accurate ASR model with a limited amount of speech data?

1.3 APPROACH

A small-scale hybrid automated speech recognition (ASR) system will be developed and studied to test if a small-scale ASR model can produce similar results to large scale models. A small transformer-based architecture will be designed and trained to predict text based on audio data received. The model will be trained on publicly available datasets such as the LibriSpeech dataset and the Mozilla Common Voice dataset. The model performance will be evaluated by using the word error rate (WER) and character error rate (CER) for the predicted text when compared to labelled text from the test datasets.

In this research, the decision to train the model from scratch without utilizing pre-training or fine-tuning with pre-existing models was deliberate. By opting for a new model architecture with training from randomized weights and biases, the focus was on creating a model tailored to the specific objectives of the study. This approach minimizes reliance on computational resources, as the model is retrained with new data without the need for extensive fine-tuning processes typically associated with pre-trained models.

In addressing the challenge of achieving results in smaller ASR models compared to current state-of-the-art end-to-end models, an alternative approach was pursued in this research. While the established methodology typically involves using model compression algorithms to reduce the parameters of a trained model to the target size, this approach was not considered here. Instead, the focus was on maximizing efficiency within available resources, particularly GPU memory constraints. The strategy involved iteratively refining the model architecture to achieve the smallest feasible size while maintaining satisfactory performance. This approach diverges from traditional model compression techniques but was deemed appropriate given the specific constraints and objectives of the research.

1.3.1 Automated Speech Recognition Framework

A basic framework is developed for an automated speech recognition model to identify the different processes and the requirements needed for each of these processes to work together and create a working speech recognizer. A speech recognition model consists of a data pre-processing subsystem where the audio data is filtered, and the text used is tokenized to character classes. The following subsystem is the feature extraction section, where selected features are created from the audio data and stored. The decoder subsystem forms the core of the speech recognition model. The decoder subsystem is where a combination of neural network architectures is designed and implemented to train the speech

recognition model using the selected features and labels. The final subsystem is the post-processing subsystem. This is where the predicted classes are converted back into text for human understanding. An example of the speech recognition model framework can be seen in Figure 1.1.

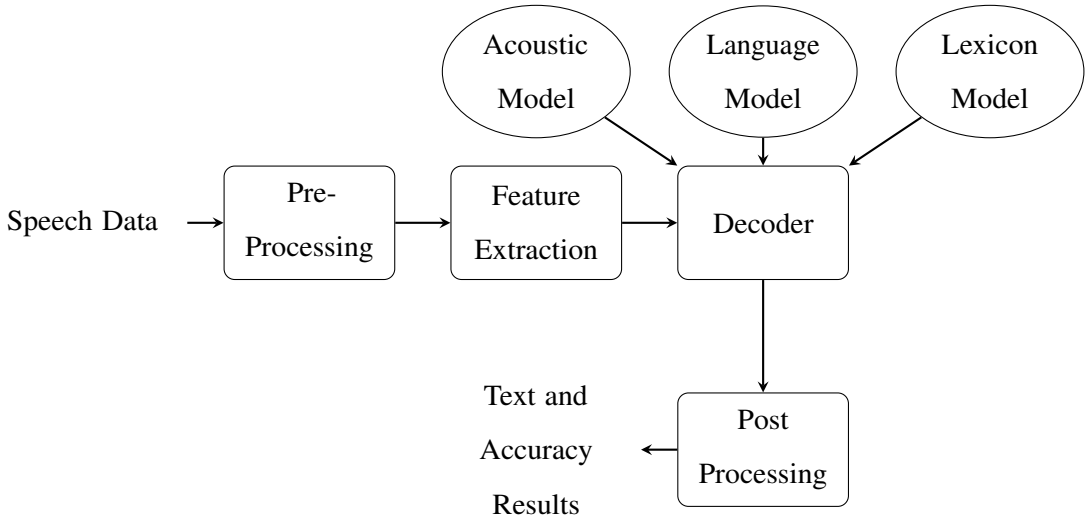


Figure 1.1. Example of a basic framework structure of an automated speech recognition model. The structure consists of a pre-processing subsystem, a feature extraction subsystem, a decoder subsystem where the neural network training occurs and finally a post-processing subsystem.

To create an automated speech recognition system using the methods discussed above, different libraries will have to be used to implement different models. Python includes multiple different audio processing libraries that can be used for cleaning the data and creating a neural network architecture. Python includes libraries such as 'torchAudio' and 'pyAudioProcessing' to extract voice data out of audio segments. For the proposed research, Python will be selected as the main programming language, as there are a substantial number of libraries available for audio processing and machine learning.

1.3.2 Available Datasets

In order to train an ASR model, sufficient training and testing data are required. Two publicly available datasets used for the training and testing of the ASR model are known as the Mozilla Common Voice dataset [18] and the LibriSpeech dataset [19]. The data is divided into training, validation and testing data. The total data consists of about 3000 hours of English speech with different dialects, which is limited, as most larger ASR models are pre-trained with over 53 000 hours of data [20]. In modern Speech Recognition evolution, 3000 hours of data can be considered as a small dataset as most of the modern models and pre-trained or trained on 40 000 hours and more of speech data. This might be

labelled or unlabelled, but our research is based purely on total available data. English was used as a proof of concept as other data was more difficult to work with due to restricted access and language understanding. If we prove that it works for English it provides confidence that it will work for other languages without any pre-training or fine-tuning from existing models. The data will be used to train and test the model. The validation data will be used to tune hyperparameters during training and to prevent overtraining.

The publicly available Mozilla Common Voice English 7.0 dataset [18], consists of 2015 hours of validated English spoken voice data from different people with different sex and dialect. The dataset used, had speech audio from 75879 different voices, but the data is a bit skewed as 45% of the data was male, 15% female and 40% unknown. The data consists of 23% United States English, 8% England English and 7% India and Asian English. The rest of the data are from other countries in the world. About 1% of the data is from Southern Africa. The data consists of 24% speakers between the ages of 19 and 29, 13% speakers between the ages of 30 and 39, 10% speakers between the ages of 40 and 49, 9% speakers over the age of 50 and 6% speakers under the age of 19. The rest of the data was produced by speakers of unknown ages. The original data is sampled at 48kHz, single channel and mp3 formatted. The Common Voice data consists of small segments of spoken speech data as the input data. The labels of the input data are the actual text extracts that were read. The Common Voice dataset has 464396 train audio segments, 16284 validation audio segments and 16284 test audio segments.

The LibriSpeech [19] dataset consists of approximately 1000 hours of read English text data that was derived from audiobooks. The data consists of different speakers consisting of male and female voices with different accents and dialects. The data has been pre-processed into a clean segment and a noisy segment. The clean segment of data consists of 464 hours of training data, 5.4 hours of development data and 5.4 hours of testing data. The other noisy data consists of 497 hours of training data, 5.3 hours of development data and 5.1 hours of testing data. The data is summarised in Table 1.1.

The LibriSpeech data consists of the spoken audiobook speech data as the input data. The labels of the input data are the actual text extracts from the audiobooks. Small segments of the data are used at a single training instance to improve data synchronisation and alignment.

The datasets require training, validation and testing sets to ensure that the ASR model being created is

Table 1.1. Data subsets in the LibriSpeech dataset

subset	hours	per-spk minutes	female speakers	male speakers	total speakers
dev-clean	5.4	8	20	20	40
test-clean	5.4	8	20	20	40
dev-other	5.3	10	16	17	33
test-other	5.1	10	17	16	33
train-clean-100	100.6	25	125	126	251
train-clean-360	363.6	25	439	482	921
train-other-500	496.7	30	564	602	1166

generalised for more speech data than just the given training data. If the model is tested on the data it was trained with, the model will seem to be very accurate, but might produce unsatisfactory results for any data outside the training dataset. The validation dataset is used to ensure that the model does not overtrain for the given training data [21].

1.3.3 Model Selection

The selected model will mainly consist of a small transformer architecture. The model receives audio data features such as MFCC values of Mel frequency values that were extracted from the sampled audio. The transformer model will be tested with different encoder and decoder sizes as well as different transformer head layers within allowed limitations. The ASR model will also require a small CNN layer to transform the input feature data into multi-layered neurons for a better exploiting of data features. The CNN layer will act as the first step of the acoustic model by mapping the audio features, while the transformer layer will act as a language model to learn the context of the sequential audio data. Python will be used to train the ASR model using a PyTorch backend. This process is required to be relatively simple due to the vast amount of data that must be processed during training and the goal of keeping the model small-scaled for low computer resource cases. The main recipe models that will be focused on are transformer models.

Different architecture layers will be combined and tested using the Common Voice data and the performance metrics such as CER and WER for each created ASR model will be compared. Once the best available model has been selected, based on the performance metrics in Section 1.3.4, slight

adjustments can be made to hyperparameters until an ideal ASR model is created. The model will be compared to other existing models using the LibriSpeech dataset and the same performance metrics. The larger the models become, the longer the training time will be and the larger the model parameter count will be. This introduces limitations as the model is only trained on a single GPU (graphics processing unit) device with 12 GB of VRAM (video random-access memory), therefore there might be difficulties in testing large ASR models [22].

Different language models are added to the end-to-end character based ASR model. This will decrease the word error rate of the models, as the predicted characters can be grouped into known English words. Different language models will be used such as the bidirectional encoder representations from transformers (BERT) language model [23] and a traditional 4-gram language model [13]. Language models could increase the word error rate of the models if it predicts words that are correct in lingual terms, but not what was said in the audio input data.

1.3.4 Performance Metrics

The different implemented ASR models will be trained and tested using the Common Voice and LibriSpeech data. The models are then compared to ASR models from literature. ASR models can be evaluated using many performance metrics. Different metrics might produce different results for the same model. All the models and different implementations mentioned above will be trained and evaluated using the following performance metrics.

1.3.4.1 CTC loss

Connectionist temporal classification (CTC) from [16] determines the maximum likelihood label for each input value. A blank token is added to the character classes to ensure repeating characters remain repeating characters. The string of characters including the blank are the possible labels for each input. The maximum likelihood labels for each input value are compressed by removing all repeating characters without a blank token and finally removing the blank classes. This set of characters is the model output and is compared to the corresponding text labels. The difference between the model output characters and text labels is known as the CTC loss. The CTC loss is used to train the model and adjust the weights and biases of the model to minimise CTC loss. The CTC loss uses the label error rate (*LER*) measure for the training and evaluation of a model. Given the *LER* of the CTC classifier h_{ctc} as the mean normalised edit distance between the classifications $\mathbf{x}$, and the targets $\mathbf{z}$, on a given dataset S_{dat} so that,

$$\mathrm{LER}\,(h_{ctc}, S_{dat}) = \frac{1}{|S_{dat}|} \sum_{(\mathbf{x},\mathbf{z})\in S_{dat}} \frac{ED(h_{ctc}(\mathbf{x}), \mathbf{z})}{|\mathbf{z}|}, \tag{1.1}$$

where *ED(a,b)* is the edit distance between two sequences a and b. The edit distance is the minimum number of substitutions, insertions and deletions required to chance sequence a into sequence b. The classification characters, **x**, and the target characters, **z**, are used as the two sequences for calculating the edit distance.

1.3.4.2 Word error rate and character error rate

The WER and CER are versions of classification accuracy performance metrics and are used for ASR models as normal classification will not represent an accurate representation of the model results. To determine the word error rate of the predicted and actual words in a sequence,

$$WER = \frac{S_n + D_n + I_n}{N_w} = \frac{S_n + D_n + I_n}{S_n + D_n + C_n}, \tag{1.2}$$

where S_n is the number of substitutions, D_n is the number of deletions, I_n is the number of insertions, C_n is the number of correct words, N_w is the number of words in the reference sequence, and

$$N_w = S_n + D_n + C_n.$$

The CER is calculated the same as the WER in equation 1.2, except that C_n is the number of correct characters instead of words. The entire reference sequence is still used, but spaces are removed. CER is a good indication of correct class identification for a character based ASR model, but the WER will give a better indication of contextual accuracy of the ASR model [24].

The performance metrics are used to evaluate the different models implemented using the LibriSpeech data. The performance metrics are the same performance metrics used in the literature and therefore the implemented models can be compared to the existing literature models.

1.4 RESEARCH GOALS

The research has the main goal of attempting to minimise the size of an ASR model, using modern architectures while limiting the reduction in performance to acceptable levels. This model is suitable for cases where there is no access to industrial scaled datasets and a lack of computer resources. A small-scale hybrid automated speech recognition (ASR) system will be studied to test if a small-scale ASR model can produce similar results to large scale models. A small transformer based architecture will be designed and trained to predict text based on audio data received. The model will be trained on publicly available datasets such as the LibriSpeech dataset [19] and the Mozilla Common Voice dataset

[18]. The model performance will be evaluated by using the word error rate (WER) and character error rate (CER) for the predicted text when compared to labelled text from the test datasets. The model performance is compared to the existing ASR models performances. Another goal is to determine if it is possible to train a small-scale ASR model using a limited amount of speech data and still acquire satisfactory performance from the model. This will provide as a proof-of-concept ASR model for under resourced languages that do not have sufficient speech data.

1.5 RESEARCH CONTRIBUTION

Modern ASR models are extremely large and consist of hundreds of millions of parameters with over a billion parameter language models added such as the model from [20]. There is limited research on small-scale ASR models with modern architectures and how they compare to the larger ASR models. The novel model that is presented in this work, consists of a small two layered CNNs that receives audio MFCC and delta features as an input. The CNN output is then transferred to a small two headed, dual layer transformer architecture for contextual training. The output of the model provides a character based prediction instead of a word based prediction to decrease the model output classes. The characters are then inserted into a small 4-gram language model to produce a more accurate word error rate. The small-scale architecture produces results that are comparable to much larger ASR models such as the gated recurrent unit (GRU) and Wav2Vec2 models from [25] and [20]. This is possible due to transformer architectures requiring less parameters to produce the same accuracy results. The model is trained on a single graphics card, due to more processing power being available for computations on local single devices, than in previous years, as well as the simplification of ASR model architectures.

1.6 RESEARCH OUTPUTS

The following article was submitted to a peer-reviewed and ISI accredited journal.

- A. Loubser, A. De Freitas, and P. de Villiers,“End-to-end automated speech recognition using a character based small scale transformer architecture.” To be published. Accessed: Apr. 18, 2023. [Online]. Available: https://dx.doi.org/10.2139/ssrn.4290605

1.7 OVERVIEW OF STUDY

Chapter 2 describes the review of the literature of traditional and modern ASR models and how they are designed, created and trained. The model performances are also compared. In Chapter 3 the research methodology is discussed. In this section the design methodology and architecture of the small-scale ASR model is described in detail. The different experiments to test the small-scale model

is also described. Chapter 4 outlines numerical and experimental results. The results mainly consist of the performance metrics mentioned in Section 1.3.4 for the different versions and training datasets of the small-scale ASR model. In Chapter 5 results are summarised and discussed. The designed model is compared to existing traditional and modern ASR models. In Chapter 6 concluding remarks are given.

CHAPTER 2 LITERATURE STUDY

2.1 CHAPTER OVERVIEW

This chapter presents the literature study for this dissertation. Section 2.2 discusses the different data pre-processing techniques for traditional and modern ASR models. In Section 2.3 traditional ASR models with separate acoustic and language models are discussed. Section 2.4 discusses modern end-to-end ASR models and transformer-based models.

2.2 AUDIO EXTRACTION AND FILTERING

2.2.1 Phoneme extraction and alignment

Creating an acoustic model requires a sequence of acoustic input vectors. These acoustic input vectors are formed from speech. Syllables are created using a human's mouth and vocal cords. When looking at spoken words, a syllable contains a vowel sound and some surrounding consonants. These syllables are added together to form words consisting out of letters. The same letters can be pronounced differently, and each pronunciation is a different sound.A phoneme is any perceptually distinct units of sound in a specified language that distinguish one word from another. The American English language has 44 phonemes, but this excludes names and places [26]. In our case we correspond phonemes with pronunciations and sounds making up letters and words. The phonemes can be divided into frames and be compared to frequency values received from an audio file that has been separated into different frames. In [26], a comparison was performed between phoneme-based speech detection and word-based speech detection. The word identification for clean speech, using a Gaussian mixture model (GMM), produced a detection accuracy of 97.9%. The detection accuracy for phoneme detection, using the same data and model was 98.3%. This slight improvement for clean speech data, increases for noisy data. The speech data has to be correctly aligned to the phoneme labels for accurate training of speech detection to occur. In [27], phoneme alignment was tested for a French acoustic hidden Markov model (HMM) using the k-fold cross validation technique. The maximum likelihood, maximum a

posteriori and maximum likelihood linear regression estimates were tested for data alignment. It was found that the maximum a posteriori estimate performed better for speech to phoneme alignment where the speech contained different accents, while the maximum likelihood linear regression estimate produced better results for clean speech where the accents were similar. In [28], a boosted maximum mutual information (MMI) function to improve discriminative training results for phoneme detection was used. The method cancels any of the shared parts of the numerator and denominator statistics for each frame of the input audio data. The boosted MME reduced the WER from 20.5% to 20.1% for 700 hours of English broadcast news data.

2.2.2 Audio extraction into features

The first step in audio extraction is to represent an audio file as a time varying signal. To detect the selected features such as the frequency of the audio signal, the signal is divided into frames [29]. The reason for sampling is that the audio signals in the time domain are represented as continuous signals and modern digital signal processing techniques need to be applied to the audio signal, requiring sampling of the signal. This in theory changes the data from analogue form to digital form. Different features can be extracted from the sampled signals.

In [29], the linear predictive coefficient (LPC), fast Fourier transform (FFT) frequency samples and Mel-frequency cepstral coefficients (MFCC) were extracted from speech data. The different features were used to train a hidden Markov model (HMM). The performance of the HMM trained on different features was tested and compared. The LPC features based HMM produced an accuracy of 79.5% while the FFT features based HMM produced an accuracy of 89% and the MFCC features based HMM produced an accuracy of 92%. The MFCC features also reduced the dimensionality of the feature vector to the model when compared to the LPC and FFT features. The derivatives or deltas of the MFCC features were added in [17]. This increased the training accuracy considerably for different speakers as the change in frequency was considered and not just the frequency of the audio signals. In [6], different audio features were tested for a supervised deep neural network (DNN) acoustic model. Using the CCLR speech dataset from Chinese television, perceptual linear prediction (PLP), MFCC and filter bank features were tested. The WER of the PLP features was 24%, while the WER for the MFCC features was 23.5%. The WER of the filter bank features was found to be the best at 22.7% for the ASR model. Raw waveform signals with an added CNN layer were compared to MFCC features for an acoustic model in [30], and it was found that the raw waveform data with an added CNN outperformed the standard MFCC features by 5%. Perceptual Linear Prediction (PLP) is calculated

very similarly to the MFCC method. The only differences are that equal loudness per-emphases is added in the filtering of the frequencies and cube-root compression is used instead of log compression. MFCCs produce slightly better results [31].

2.3 TRADITIONAL AUTOMATIC SPEECH RECOGNITION MODELS

Using a mixture of deep learning and natural language processing (NLP), a speech to text model can transform audio into text as well as predict the highest probability text that belongs to the audio section. This is in essence what automatic subtitle generation consists of [4]. ASR systems can be created using two main models. The acoustic model represents the mapping or connection between the audio features and phonemes. The language model represents the probability of dependency between letters in a word or words in a sentence. Both these models require a very large amount of training as well as very large vocabulary and phoneme libraries [22]. The acoustic model must be able to ignore sounds that are not speech. The language model should be able to separate one word that has the same phonemes as other words by looking at the previously predicted words. The model needs to be able to distinguish between ‘bear’ and ‘bare’ using context training. This requires a significant volume of text training data as well as a large vocabulary library [32].

2.3.1 Acoustic Modelling

Traditionally, ASR models consisted of separate acoustic and language models. Large acoustic models such as [5] and [6], consist of very large deep neural networks (DNNs) that use large hidden Markov models (HMMs) and feedforward neural layers to represent the mapping between phonemes and the audio signals or features. The acoustic model architecture presented in [5] was a deep neural network (DNN) with multiple hidden layers, low-rank approximation to the final weight matrix and many contexts dependent hidden Markov model (HMM) states for large ASR datasets. The model was trained using 10 million words from the DNTrain YouTube database. A GMM, with 18000 possible output states, was trained using the same data and tested with noisy YouTube data and produced a WER of 52.3% which is not very accurate. The DNN model with 7 hidden layers and 7000 possible output states, produced a WER of 44% on the same data. The DNN model with 7 hidden layers and 45000 possible output states, produced a WER of 42.9% on the same data, but required significantly more training time. Increasing the hidden layer states to 14, produced a WER of 40.9%. The model did not produce very accurate results as the test data from YouTube was very noisy, but the more parameters the model had, the better the results that were produced.

Clean speech data was used in [7] to train and test a DNN acoustic model and produced a WER of

13.4% for speech and 13.1% for music. The cleaner the speech data, the more accurate the results were. In [6], clean speech data from Chinese television was used to train a DNN acoustic model that was combined with a GMM. The model produced a WER of 20.8% of the Chinese speech data compared to 24.2% using a normal GMM model. Multiple DNN models can be combined to produce better results. In [8], two DNN models were used for speech recognition. The first model was used for noise reduction of the speech data. The second model was the acoustic model that used the output of the first model as input data. A baseline GMM-HMM produced a WER of 25.7%. The acoustic model produced a WER of 16.77% and the combined model produced a WER of 10.76%.

2.3.2 Recurrent Neural Networks

Recurrent neural network (RNN) models and long short-term memory (LSTM) models [12] made a significant impact on ASR, as the models could be trained to take the context of the speech into account by training the model with sequential speech data [7]. RNNs contain feedback loops that allow previous information to persist in future calculations. Long term dependencies for data connection can be a concern in neural networks. If context is derived from a large range of data as the case might be in subtitle generation, normal RNNs might struggle to connect the data. LSTMs are a class of RNN algorithms that can handle long term dependencies [11]. In [12], an LSTM algorithm is compared to an RNN model and a FFNN model. The dataset used, consists of an English language model corpora of 3.1 billion words which was reduced to 50 million words for training. The data has a vocabulary of 150 000 different output states. The FFNN produced a WER of 11.3%. The RNN produced a WER of 11.1% and the LSTM produced a WER of 10.4%. A fusion of a dual LSTM model and a coupled HMM was developed in [22] and was tested using an English dataset and French dataset separately. The English dataset produced a WER of 17.45% compared to a basic GMM-HMM WER of 39.22%.

2.3.3 Language models

Statistical language modelling is a crucial part in ASR, as it makes the predicted text understandable for a human by putting words in context with the sentences. The models also generally decrease the word error rate for ASR dramatically. Recurrent Neural Network Language Models (RNNLMs) provide a powerful and efficient method to model sequential data [33]. The first language models consisted of *n*-gram models and HMMs [13]. The *n*-gram models and HMMs could be trained with significantly less computation effort when compared to RNNLMs but did not produce as accurate results as the more modern and larger RNNLMs. As more computational processing power became possible as in [9], neural network models replaced *n*-gram models and HMMs. In [34], smaller n-gram

language models, that were smoothed out to remove uncommon words were produced. This created small n-gram language models that produced similar results to original n-gram language models, but at double the speed and half the memory.

RNNs were initially tested for acoustic and language modelling, but the results obtained for the amount of processing power and memory required was undesirable when compared to *n*-gram language models. This has changed due to computers being able to process data faster and more efficiently. RNNs are the preferred models over *n*-grams when it comes to large language models [13].

A language model with an RNN was tested and compared to an *n*-gram language model in [10]. Training data consisted of 8 billion words from three different sources: 880 million words from spoken news, 1.7 billion words from Wikipedia and 6.1 billion words from web crawling popular sites. Different final state (*n*-state) sizes were used. Increasing the *n*-state, increased the neural network size and produced a better perplexity result, but required more training. Perplexity is another performance metric for language models where it measures the understanding of words followed by one another. For a standard 5-gram model using the given data in [10], the perplexity was 66.9%. The tested RNN with 2048 states produced a perplexity of 45.2% and the RNN with 4096 states produced a perplexity of 42.4%. The accuracy of the models are drastically improved using RNNs, but at a very large computational cost. In [4], an RNN is used with GPU training to increase the training speed. This makes large RNN architectures for ASR possible. The GPU trained RNN produced a WER of 15.16% on a 20 million words dataset. The same model, trained on a CPU, produced similar results, but at a training cost of 53 times more time required than the GPU trained model. In [12], LSTM language models are shown to perform better than n-gram language models and RNN language models. Meaningful contextual information is the main required output for a language model.

A context-based language model was created in [35], that changes context available based on the area of speech that is being used. This would be for a use case where the speech data topics are previously known. Out of vocabulary (OOV) words were also considered, so that the context dependent language model would still allow predicted words outside the language model vocabulary. This method reduced the WER by 44% for the specific use cases, when compared to a large general language model. Adaptive neural language modelling was attempted in [36], where the input and output layers of the language model is changed based on the input representations and limited words. The original language model used had 4371 million parameters. This was reduced to 1026 million parameters for

a large language range model and 331 million parameters for a small language range model. The small language range model produced similar results to the larger language range model, when a small dataset was used. Multiple language models were decreased in size by using factorization and compression of the output layers, based on the required language and word set being used. This caused the increase in computational speed of language models and allowed language models to be added as an extra word identification layer after end-to-end ASR models to create a complete ASR system [37].

2.3.4 Selected Recurrent Neural Network Models

In [33], a combination of two singleton recurrent neural fuzzy networks (SRFRNR) are implemented to train a speech recognition model with noisy data. The SRFRNR is compared to a multilayer perceptron (MLP) model and a time-delay neural network (TDNN) model. The first SRFRNR model is used for noise filtering and the second SRFRNR model is used for speech recognition. At a sound to noise ratio (SNR) of 24 in the training and testing data, the SRFRNR model produced an accuracy of 85% compared to the other models' accuracy of between 70% and 78%. Different datasets provided WERs between 15% and 45% for the RNN language models tested in [38]. The noisier the data, the larger the WER.

The model in [25] uses CTC-based training for end-to-end ASR and is tested on the LibriSpeech dataset. The model consists of two convolutional layers and 5 GRU layers with 52.5 million parameters. The model produces a WER of 11.9% and 31.1% on the test-clean and test-other dataset respectively. An added 4-gram language model produces a WER of 8.3% and 24.4% on the test-clean and test-other dataset respectively. [39] also uses an older LSTM based architecture to create a character-based ASR model.

2.4 MODERN AUTOMATED SPEECH RECOGNITION MODELS

The introduction of attention-based models such as transformers from [14], allowed ASR models to be more efficient and deliver the same results with less computational power required. Modern end-to-end speech recognition systems cut out the phoneme step to reduce the complexity of the system and to reduce error propagation. Transformer architectures are state-of-the-art replacements for LSTM architectures. Transformers are memory efficient, fast neural architectures that have reduced parameters and increased training speed when compared to RNNs [40]. In [41], a Conformer is introduced, an architecture that combines CNNs and Transformers for ASR tasks. This model achieves state-of-the-art accuracies on the LibriSpeech benchmark, outperforming previous Transformer and

CNN-based models by effectively modelling both local and global dependencies in audio sequences. Our approach is similar but uses a CNN architecture before the transformer architecture, while [41] builds the CNN inside of the transformer and the architecture is noticeably bigger in parameter size. Our approach uses MFCC audio features while the approach in [41] uses raw audio. In [42], a low rank transformer was used for an end-to-end speech recognition system with reduced parameters and increased training speed and inference. The model was based on mandarin speech and produced a CER of 13.6% for an 8.7 million parameter model without an added language model on the mandarin AiShell-1 dataset. An RNN model produced a CER of 19.43% and had exponentially more parameters than the low rank transformer. [1] and [43] produce character-based ASR models with an added language model. The models vary and have up to 48 layer transformers.

2.4.1 Transformer based language models

Transformers improved language models drastically as larger models could be trained more efficiently [44]. Bidirectional encoder representations from transformers (BERT) models were one of the first large transformer-based language models along with generative pre-trained transformer (GPT) models. In [23] a multi-billion-word corpus was used to train a BERT language model that achieved a general language understanding evaluation (GLUE) score of 80.5%. This was a 7.7% increase when compared to any previous work. The BERT model can be fine-tuned using transfer learning on the last layer for ASR word correction use cases. The GPT-3 language model is a 175 billion parameter model that is the state-of-the-art transformer-based language models [45]. The model is too large and complex for purely ASR. The language model is currently used for automated article and paper writing. Transformer-based language modelling and decoding produced the ability to use lattice rescoring and predict text, based on a longer range of history in the sequential text data. A weighted average method is used to determine the importance of given words given their relative positions in sequences of words. In [46] the ‘XL-net’ language model is researched. The model consists fewer parameters than the BERT model, but still has over 100 million parameters, while the BERT model has 110 million parameters for the small model and over a billion parameters for the large model. Both the transformer-based language models produce very similar results, but GPU memory availability caused an issue while running an ASR model with an added transformer based language model. An older RNN based language model consisting of 15 million parameters produced very similar results on the ‘TED-LIUM’ dataset. The BERT model was compressed in [47] using a compressed transformer model, while still producing the same results. The compressed version of BERT is named ‘huBERT’ and still has between 90 million parameters and over a billion parameters depending on the smallest and largest ‘huBERT’ model.

In recent research, there has been notable exploration in enhancing Automatic Speech Recognition (ASR) systems leveraging LLMs and knowledge transfer techniques. In [48], the integration of LLMs into ASR systems was investigated to improve transcription accuracy. The study, utilizing the Aishell-1 and LibriSpeech datasets, evaluated the potential of employing LLMs' in-context learning capabilities. Despite initial experiments resulting in higher WER, the study shed light on the challenges of effectively leveraging LLMs for ASR applications. Another approach presented in [49] addresses the data hunger challenge in training end-to-end ASR systems by transferring knowledge from pretrained language models. The paper proposes a method to transfer semantic knowledge from embedding vectors of large-scale language models to improve ASR decoders' performance without added computational costs during decoding. The research in [49] is attempting to improve speech recognition by improving the language context with the use of LLMs.

In [50] textless NLP is attempted to process language directly from the input speech units, but this would give the speech understanding without text involved which would defeat the purpose of text in an ASR environment. The model was used in conjunction with text based ASR models to provide WERs that were comparable to standard older ASR models. In [51], a BERT model was directly used for a mask based ASR model to predict text from raw audio. The linguistic characters from raw audio were used as the input to the BERT mask based model and phoneme styled words were expected as an output. Unfortunately, the model only produced CERs and WERs that were comparable to smaller and older ASR systems. The BERT model uses a masked input training method, where one data input of the sequence is masked, and the rest of the input sequence is used to predict the masked data input value. In [51] a simple acoustic model was stacked on the pre-trained BERT model to reduce error propagation, but produced CERs of between 54.6% and 99%, which is not satisfactory results for an ASR model. Fairseq [52], is a fast and extensible sequence modelling toolkit for translation, summarization and language modelling using transformers. The models were trained on an industrial 100Gb VRAM graphics card with a billion-word corpus. This toolkit was extended, and an ASR model was created in [53].

2.4.2 Transformer based ASR models

Multiple end-to-end transformer models are compared in [54], using the LibriSpeech dataset [19]. The paper shows that transformer models are superior with a small volume of speech data, but as the data volume is increased all models converge to the same results and rely less on language models for accurate results. The transformer model with 322 million parameters and using CTC training, achieved

a WER of 2.99% and 7.31% on the dev-clean and dev-other data respectively. The transformer also achieved a WER of 3.09% and 7.40% on the test-clean and test-other data respectively. Adding a 4-gram language model increased the model accuracy slightly to a WER of 2.86% and 6.72% on the test-clean and test-other data respectively. A Fairseq toolkit fast speech to text transformer model is used in [53], that has 263 million parameters and has an added transformer-based language model that was pre-trained on the 'Wikitext' text corpus of over a billion words. The model produces a WER of 3.3% and 7.7% on the test-clean and test-other LibriSpeech dataset respectively.

Modern very large ASR systems such as [55], produce lower word error rates, but are very large and take significant computational power to train and run. In [55] and [56] the Wav2Vec ASR model, consisting of mainly transformer-based architectures, is used for speech recognition. In [56], the model outperforms the previous best character-based ASR model [57] with a WER of 3.2% using the Wall Street Journal dataset, but at a significant reduction of training data. The Wav2Vec model still uses over 53 000 hours of data, where the model in [57] used over 100 000 hours of speech data. In [20] it is found that a model can be trained with as little as 10 minutes but states that "Using just ten minutes of labeled data and pre-training on 53k hours of unlabelled data still achieves 4.8/8.2 WER". The model in this research focuses solely on labelled data without any pre-training or fine tuning as it is a new model. This is a small scale direct classification model without pre-training to understand general speech representations as this requires a lot of computational power and large amounts of original data. A WER of 3.8% and 6.5% on the LibriSpeech test-clean and test-other data is achieved by [55], that uses a very large transformer-based ASR model of over 300 million parameters called the Wav2Vec architecture and a BERT language model that was fine-tuned with 53 000 hours of 'Libri-Light' text data. [20] achieves a WER of 1.8% and 3.3% on the LibriSpeech clean and other test set. The model is trained using 53 200 hours of audio. The model contains 24 transformer blocks of dimension 1024 and inner dimensions of 4096 and 16 attention heads. The model is accompanied by a transformer-based language model. The large model has 317 million parameters without the language model. Including the language model it is over a billion parameters. In [58], a cross-lingual speech representation model called XLSR is created from the wav2vec2.0 model from [20]. The existing model is fine-tuned with Common Voice data [18] consisting of 53 languages and other multilingual datasets. The larger amount of training data reduces the WER drastically when compared to other models. The model produces an average WER of English Common Voice testing data of 7.6% over 53 languages with a 4 gram language model. Using the Common Voice en 7.0 dataset in [59] the Wav2Vec2 XLSR model trained on 53 000 hours of data produced a WER of 27.72% and a CER of 11.65% for the testing dataset.

Adding a language model produced a WER of 20.85% and a CER of 11.01%.

A new transformer model named XLS-R, [60], produces a 2 billion parameters, transformer-based, ASR system that is trained on 436 000 hours of data from over 128 languages. The model outperforms all previous single language models for languages that have a small volume of training data available. For English the Wav2Vec2 model outperforms the new XLS-R model when using the LibriSpeech data for comparison by a WER of 5.6% compared to 5.9% using the test-clean dataset. XLS-R can be fine-tuned with as little as 10 minutes of labeled training data after being fine tuned with the initial 436 000 hours of data, resulting in competitive Word Error Rates (WERs) when coupled with external language models. This aspect highlights the model's adaptability and efficiency in scenarios where labeled training data is scarce. The model would still require the initially trained parameters to achieve accurate results with limited audio. Once the model is initially trained it is also possible to train the model using unlabelled data using self-supervised fine tuning languages without labelled data [60].

All the character-based models mentioned above are trained using the CTC labelling method. In [16], the CTC method of classifying and labelling unsegmented data is researched and discussed. CTC uses a forward-backward algorithm with maximum likelihood training during the ASR model training to maximise the probabilities of correct labelling. The method outperforms previous segmentation methods such as the hidden Markov model segmentation method. CTC decodes the input data into a known dictionary of classes which can be a set of characters or words. The CTC method can only decode data into known classes; therefore an unknown class was added in [61] to take into account any words and characters not seen before.

2.5 CHAPTER DISCUSSION

In Section 2.2 different data pre-processing techniques were explored. It was found that MFCC features, and Mel frequency features are the most accurate for speech recognition models. The MFCC features were selected due to the MFCC data features being more compressed than the Mel frequency feature data, while containing most of the significant audio data.

Section 2.3 introduces traditional ASR models that consisted out of very large acoustic models with added language models. The ASR models improved drastically with the introduction of RNNs. In [25] an end-to-end ASR model, with GRU layers and 52.5 million parameters, produced a WER of 8.3%

and 24.4% on the test-clean and test-other LibriSpeech dataset respectively. This was achieved with an added 4-gram language model. Modern ASR models in Section 2.4 consists mostly of transformer-based architectures. The end-to-end models in [54, 55, 20] have over 300 million parameters and produce WERs of between 3% and 8% for the test-other dataset of the LibriSpeech dataset. A small-scale architecture similar to the architecture in [42] will be designed to experiment if a small-scale, transformer-based architecture is comparable to the modern larger ASR models.

CHAPTER 3 METHODS AND DESIGN

3.1 CHAPTER OVERVIEW

This chapter presents the methodology, design and experiments of the small-scale, transformer-based ASR model for this dissertation. Section 3.2 provides the methods and calculations of the selected data pre-processing techniques for traditional and modern ASR models as well as the text tokenization. In Section 3.3 the small-scale ASR model is designed with a CNN architecture and a transformer architecture. The model is designed in detail and implemented with a modular approach to be able to change the model size and hyperparameters. Section 3.4 implements different language models to the ASR model, such as a traditional n-gram language model and a BERT language model. The basic design of each language model is described and how it is implemented into the ASR model.

3.2 DATA PRE-PROCESSING

The audio files are sampled and filtered for a selection of audible frequencies. Selected features are extracted that represent the speech from the sampled and filtered audio files. These features are represented as vectors with real values that a learning model can use for training. The vector values represent different frequencies which are grouped together to form phonemes. Phonemes are short audio units representing parts of characters or words. The extracted phonemes from the audio are compared to a library of existing phonemes or characters in this case by having each audio section or phoneme labelled as a character or part of a word. This is to separate the sequential features, to match the class labels of the audio data and ensuring accurate training of the model.

Creating an ASR model requires a sequence of acoustic input vectors that represent audio features. These acoustic input vectors are formed from digital recordings of known speech. The digital recorded speech is classified into two categories namely voiced and voiceless speech [26]. Voiced sound is produced by a human's vocal folds being tensed up and relaxed using air flow. This repeating cycle

produces sound at different frequencies for men and women. The fundamental frequency produced by each person is perceived as the pitch. Generally a male has a fundamental frequency around 125Hz and a female has a fundamental frequency around 210Hz [26]. This can change dramatically based on the person's vocal folds. Voiceless sounds are produced without the vibrating effect of the vocal folds. The sounds created are modulated by articulation to create different resonance sounds.

The lips, teeth and tongue are used with the speech sound for syllable creation. These syllables are then added together to form words consisting out of letters. The same letters can be pronounced differently, and each pronunciation is known as a phoneme. The American English language has 44 phonemes, but this excludes names and places [26]. These phonemes can then be acoustically realised as phones. Phones are the way phonemes are pronounced. Different phones exist for the same phonemes. The recorded digital phones received from an audio file that has been separated into different frames are labelled to the 44 known theoretical phonemes. In the case of an English character-based end-to-end ASR model, the phonemes represent characters of the English language and multiple different sounds can be classified as a single character and different characters might be predicted from the same sound. This ambiguity is resolved by inspecting the surrounding predicted characters of the audio sequence to predict the correct word.

3.2.1 Audio Data Sampling

The first step in audio extraction is to represent the audio file as a time varying signal. The signal will consist of different amplitudes based on loudness. In speech the range of a human voice is generally up to $F_{max} = 4kHz$ [17]. The sampling rate, F_{samp}, is chosen as 8kHz or 16kHz to adhere to the Nyquist sampling rate such that

$$F_{samp} = 2 \times F_{max}. \tag{3.1}$$

The reason for sampling is that the audio signals in the time domain are represented as continuous signals and modern digital signal processing techniques need to be applied to the audio signal, requiring sampling of the signal. The sampling rate allows the amplitude data of the audio signal to be stored at every 62.5us when sampled at 16kHz and every 125us when sampled at 8kHz for processing. This in theory changes the data from analogue to digital.

The original audio data from the datasets used are read into the model as a time-based waveform and resampled to either 8kHz or 16kHz. The resampling step first required the greatest common divisor

between the original sampling rate and the new sampling rate of either 8kHz or 16kHz. The resampling method uses a finite impulse response (FIR) low pass filter with a width of 6 and a roll-off value of 0.99. The resampling method used is sinc-interpolation,

$$x_{sinc}(t) = \sum_i x[i] sinc\left(\pi \times f_{orig} \times \left(\frac{i}{f_{orig}} - t\right)\right), \tag{3.2}$$

where $x_{sinc}(t)$ can be exactly reconstructed from the input $x[i]$ and f_{orig} is the original frequency divided by the greatest common divisor between the original sampling rate and the new sampling rate.

The sinc-interpolated signal $x_{sinc}(t)$ can be sampled with a different sampling rate to produce $y[j]$ using:

$$y[j] = \sum_i x[i] sinc\left(\pi \times f_{orig} \times \left(\frac{i}{f_{orig}} - \frac{j}{f_{new}}\right)\right), \tag{3.3}$$

where f_{new} is the new frequency divided by the greatest common divisor between the original sampling rate and the new sampling rate.

The newly sampled signal $y[j]$ is therefore the convolution of $x[i]$, with a finite impulse response (FIR) approximation filter stopping at the selected filter width of 6. Looking at equation (3.4) below,

$$y[j + f_{new}] = \sum_i x[i + f_{orig}] sinc\left(\pi \times f_{orig} \times \left(\frac{i}{f_{orig}} - \frac{j}{f_{new}}\right)\right), \tag{3.4}$$

it is determined that, $y[j+1]$ until $y[j+f_{new}]$ will have different weights, but will use the same filter. Therefore, $y[j+f_{new}]$ uses the same filter as $y[j]$, but on a shifted version of x by f_{orig}.

3.2.2 Audio Feature Extraction

At each sampled time instance, the signal can be converted into the frequency domain to represent the different frequencies of the audio signal at different points in time. The spectrogram, representing the frequency domain of the audio data, can be calculated from the original audio data that is separated into fixed time frame segments. To detect the selected features such as the frequency of the audio signal, the signal is divided into frames of between 20ms and 40ms. For audio signals there should be enough data or signal points in the frame period to identify any frequencies in the possible range of voice frequencies for that frame period. This is determined based on the sampling rate of the audio signal. The ideal frame size is selected to be 25ms [17]. There are therefore 400 samples in every frame when sampled at 16kHz and 200 samples in every frame when sampled at 8kHz. Each frame is selected using a frame step size. The ideal speech recognition step size is 10ms or 160 samples at a sampling frequency of 16kHz [17]. The first frame would be samples 0-400. The second frame would then be samples 160-560 and so on.

The first step is to calculate the Discrete Fourier transform (DFT) for each frame [17]. This is to extract as much frequency and time-based data from the audio files as possible. The DFT is calculated as follows:

$$S_{dft}(k) = \sum_{n=1}^{N} x_i(n)h(n)e^{-j2\pi kn/N_{samp}}, \tag{3.5}$$

where $S_{dft}(k)$ is the complex DFT for each frame, i, and each discrete frequency index k. The discrete frequency index, k, ranges from 0 to the length of the DFT which would ideally be a 512 point FFT [17]. Each FFT point corresponds to a bin. Each discrete frequency index k would be at the edges of the bins and each bin size can be determined by taking F_{samp}/FFT_{points}. With 400 sample points there would be 112 zero values added for the 512 point FFT. This is known as zero padding. $x_i(n)$ is the time domain signal for each frame i and each sample point n. For this method, the FFT was selected to be 400 points to remove the need of zero padding.

Filtering is required to gradually drop the signal amplitude at the edge of each frame. This is to reduce sidelobes in the frequency domain. The function, $h(n)$, is an N_{samp} sample length analysis window for filtering. An example would be the Hamming window or Hanning window. The formula is produced as follows [31]:

$$h(n) = (1 - \alpha_{filt}) - \alpha_{filt}cos\left(\frac{2\pi n}{L_{win} - 1}\right), \tag{3.6}$$

where L_{win} is the window length and $\alpha_{filt} = 0.46164$ for the Hamming window and $\alpha_{filt} = 0.5$ for the Hanning window. The periodogram-based power spectral estimate, $P_i(k)$, for each frame can then be calculated as:

$$P_i(k) = \frac{1}{N_{samp}}|S_i(k)|^2. \tag{3.7}$$

In the case of a 400 point FFT, only the first 200 coefficients are generally kept as they represent the frequencies from 0 to F_{max}. These values represent the frequencies between 0Hz and 8kHz if the data is sampled at 16kHz as in Figure 3.1. The number of FFT points N_{samp} was selected to be 400

to match the number of samples for a 16kHz sample rate and a 25ms frame period. The window size was selected to be the same as the number of FFT points of 400. The window hop length (L_{win}) between DFT windows was selected to be half the number of FFT points and therefore 200. The window method that was selected, is the Hanning window which returns 1 values everywhere inside the window size. The final spectrogram function returns a data output of size $(x, freq, time)$, where $freq$ is $N_{samp}/2+1$ and $time$ is the number of window hops or number of frames as in Figure 3.1.

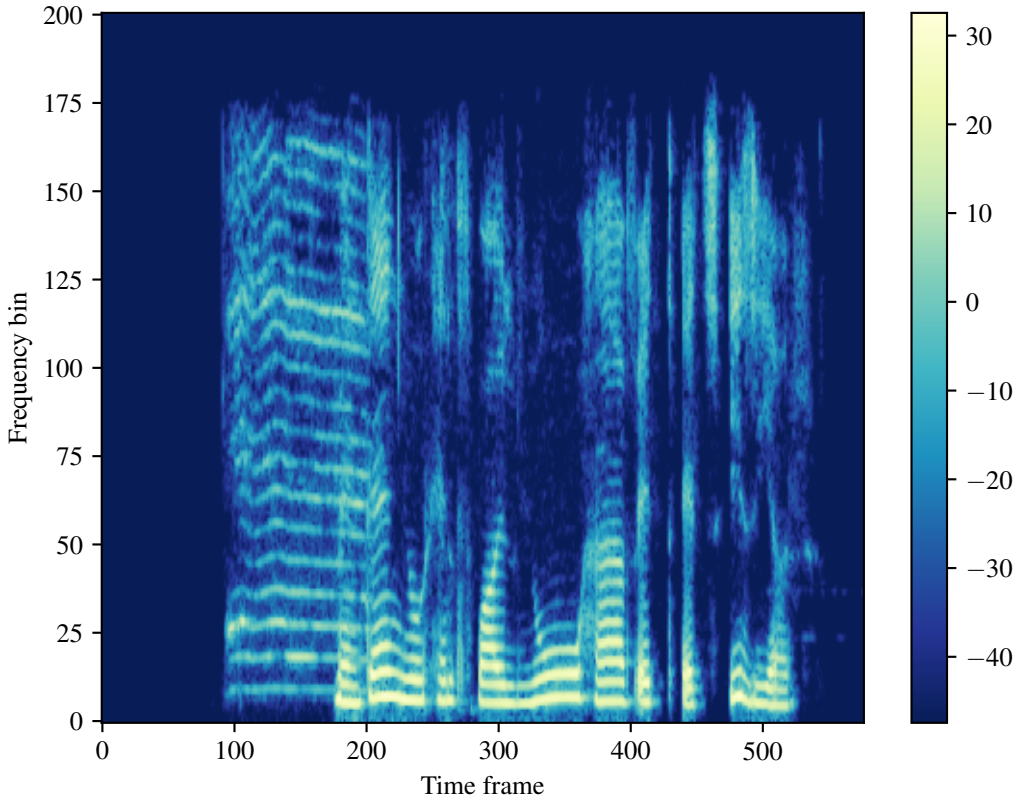


Figure 3.1. A spectrogram example of an audio track sampled at 16kHz, with a female voice saying: "Maintain your health while you have it, it's easier". The spectrogram consists of 201 frequency bins or FFT points and 577 time bins. It is noticed that the frequency amplitudes are higher in the lower bins, therefore showing that the sampled frequency covers all the vocal data.

3.2.2.1 Mel-Spectrogram

The sampled spectrogram data can be converted into the Mel-spectrogram. The Mel-spectrogram is an adjusted version of the normal spectrogram based on how frequency is perceived by the human ear. The Mel scale relates the perceived frequency also known as the pitch of audio to the actual measured or calculated frequency. Humans can interpret small changes in pitch for lower frequencies. As the

frequency is increased, the human ear can not distinguish between small pitch changes any more. The original spectrogram data is multiplied with a Mel filterbank consisting of triangular frequency ranges consisting of a preselected amount of filterbank triangles.

The Mel-spaced filterbank, which consists of triangular filters that are applied to the periodogram power spectral estimate calculated in equation 3.7, is calculated. The selected number of filters was chosen to be 81. The filterbank consists of 81 vectors of length 200 each due to the FFT size. Most of the vector values will be zero except at the frequencies where there are active audio frequencies.

To create the Mel-filterbank, the minimum and maximum frequencies are selected. This would generally be 0Hz and 8kHz, when sampled at 16kHz [17]. The bottom and top frequencies are then transformed into the Mel scale, $M(f)$, where

$$M(f) = 2595 \cdot log\left(1 + \frac{f}{700}\right). \tag{3.8}$$

Once the minimum and maximum Mel values have been calculated, there are 81 linearly spaced values taken in between, thus creating 83 Mel discrete frequency values for an 81 filter Mel-filterbank. The 83 values are transformed back to the frequency scale using

$$M^{-1}(m) = 700\left(e^{(m/1125)} - 1\right). \tag{3.9}$$

The 83 computed Mel frequency values will not exactly correspond to the 200 frequency values of the FFT. The Mel frequency values are just rounded to the nearest corresponding frequency bin index of the FFT, given the sampling rate. The FFT frequency values are calculated using the sampling rate of 16kHz and the FFT size of 512.

The actual filterbanks can now be created. Each filterbank will have a 0 at the starting point, a peak value at the next point and reach a 0 at the 3rd point as follows:

$$H_m(k) = \begin{cases} 0 & k < f(m-1), \\ \frac{k - f(m-1)}{f(m) - f(m-1)} & f(m-1) \leq k \leq f(m), \\ \frac{f(m+1) - k}{f(m+1) - f(m)} & f(m) \leq k \leq f(m+1), \\ 0 & k > f(m+1), \end{cases}$$

where m is the number of the 81 different filters and $f(\cdot)$ is the frequency value in the list of 81+2 Mel-spaced frequencies. The discrete frequency index k, is between 0 and 200. The equation above produces a frequency distribution similar to a 10 filter bank example in Fig 3.2 below.

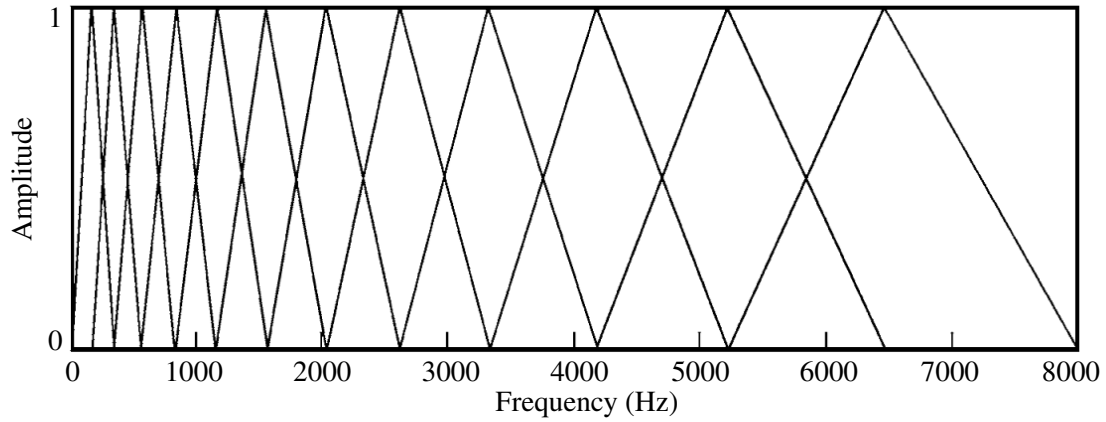


Figure 3.2. Example of a set of 10 filter banks, where each triangle represents a set of frequencies and their corresponding amplitude levels. Each triangle is known as a filterbank. Image extracted from [17].

Each Mel-filterbank is multiplied with the power spectrum and the coefficients of all 200 vectors are added together. This produces an energy value for each filterbank. The 81 energy values from the filterbanks are transformed into log filterbank values such as in Figure 3.3

3.2.2.2 Mel Frequency Cepstral Coefficient

A discrete cosine transform (DCT) is applied to the Mel-spectrogram to form the Mel frequency cepstral coefficients (MFCC). MFCCs decrease the features of the data while keeping most of the information of the original audio data. The 81 Mel features were decreased to 16 MFCCs. The DCT formula used was:

$$X_{DCT}[k] = \sum_{n=0}^{N_{Mel}-1} M_{spec}(n) \cos\left[\frac{\pi}{N_{Mel}}\left(n+\frac{1}{2}\right)k\right], \quad \text{for } k = 0,\ldots,N_{MFCC}-1. \tag{3.10}$$

The DCT, $X_{DCT}[k]$, of the Mel-spectrogram $M_{spec}(n)$ is calculated with N_{Mel} being the number of Mel features, n being an index with values ranging from 0 to $N_{Mel}-1$ and k being an index with values arranged from 0 to $N_{MFCC}-1$. The Discrete Cosine Transform (DCT) of the 81 log filterbanks produced 16 cepstral coefficients. The 16 MFCCs produced are features for each frame section of the

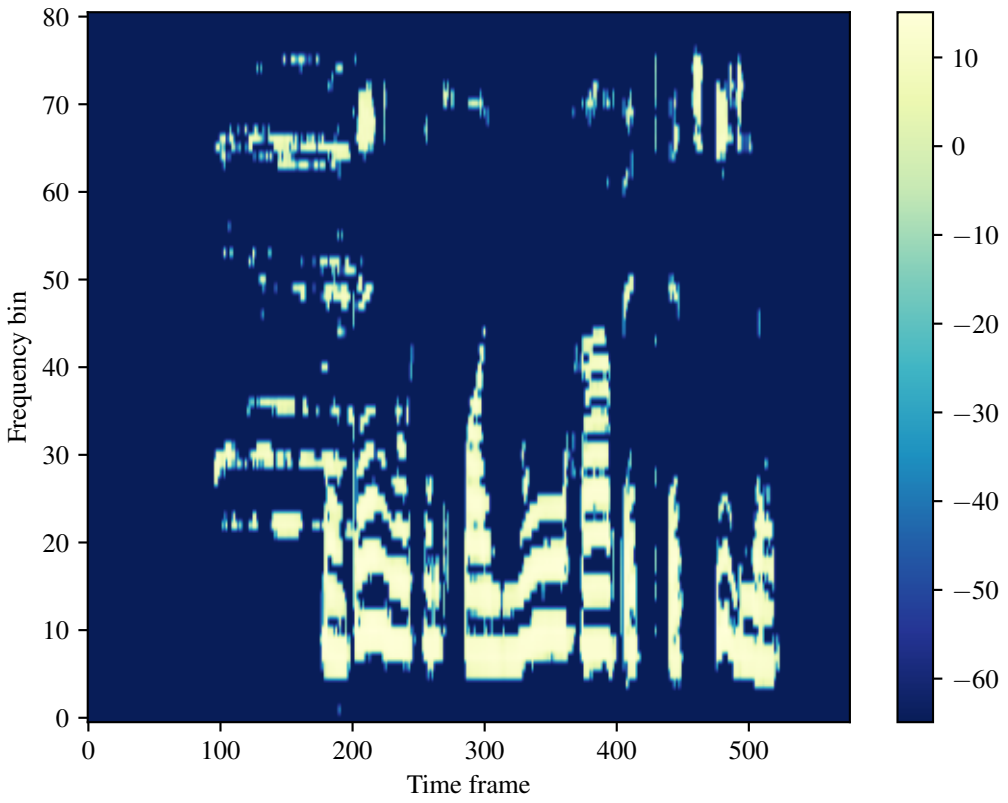


Figure 3.3. A Mel-spectrogram example of an audio track sampled at 16kHz, with a female voice saying: "Maintain your health while you have it, it's easier". The Mel-spectrogram consists of 81 frequency bins or FFT points and 577 time bins. It is noted that the vocal data is more distinguishable from the silence data, than in the normal spectrogram.

audio data. The MFCC features are more decorrelated than the Mel spectrogram which is beneficial to the linear parts of the model.

3.2.2.3 Delta Coefficients

The deltas or change in MFCC values with respect to sample time are also calculated and added. The deltas of the MFCC values were also calculated to append to the features of the audio and consider change in frequency to improve the model for multiple different speakers. The delta MFCC describes the change in MFCC values for frames. As such, the number of feature values will double, since there are 16 MFCC coefficients and 16 delta coefficients. The Delta MFCC, D_t, is calculated as

$$D_t = \frac{\sum_{n=1}^{N_{del}} n(C_{t+n} - C_{t-n})}{2\sum_{n=1}^{N_{del}} n^2}. \tag{3.11}$$

The value $N_{del} = \frac{l_{win}-1}{2}$, where l_{win} is the selected delta window length. The delta window length determines how many MFCC time samples are used to calculate the MFCC delta value. A l_{win} value of 2 was selected to take the change between two MFCC time samples. D_t is the calculated delta coefficient and C_t is the static MFCC coefficients.

The MFCC and delta coefficients were concatenated into a single matrix in Figure 3.4. The MFCC values represents the first 16 frequency bins, and the delta coefficients represent the second 16 frequency bins. The MFCC data represents the voice data in terms of frequency while the delta coefficients represent a change in frequency between MFCC bins.

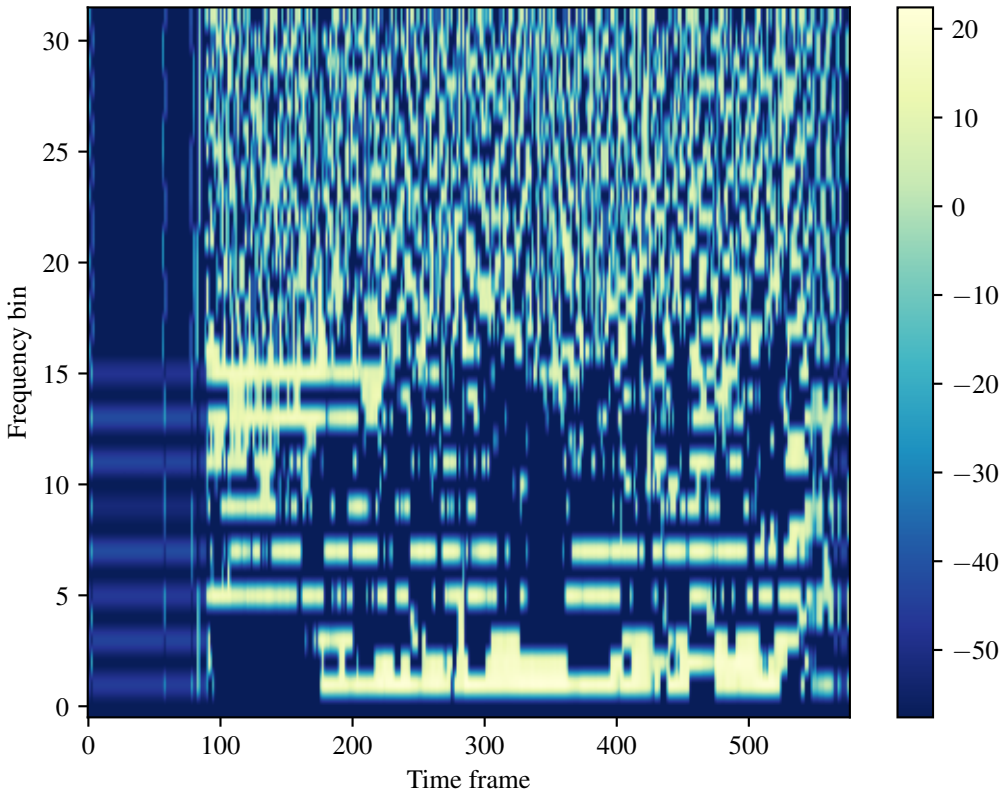


Figure 3.4. A combined MFCC and delta coefficients example of an audio track sampled at 16kHz, with a female voice saying: “Maintain your health while you have it, it’s easier”. The MFCC consists of 16 frequency bins and the delta coefficients consist of the next 16 frequency bins. There are also 590 time bins. It is noted that most of the plot is filled with usable data and the frequency features are reduced from 201 features to 32 features.

3.2.3 Feature pre-processing

Different pre-processing methods such as “SpecAugment” and time stretching is implemented to introduce random perturbations of selected audio data during training. This virtually increases training data and generalises the ASR model for more different speech types.

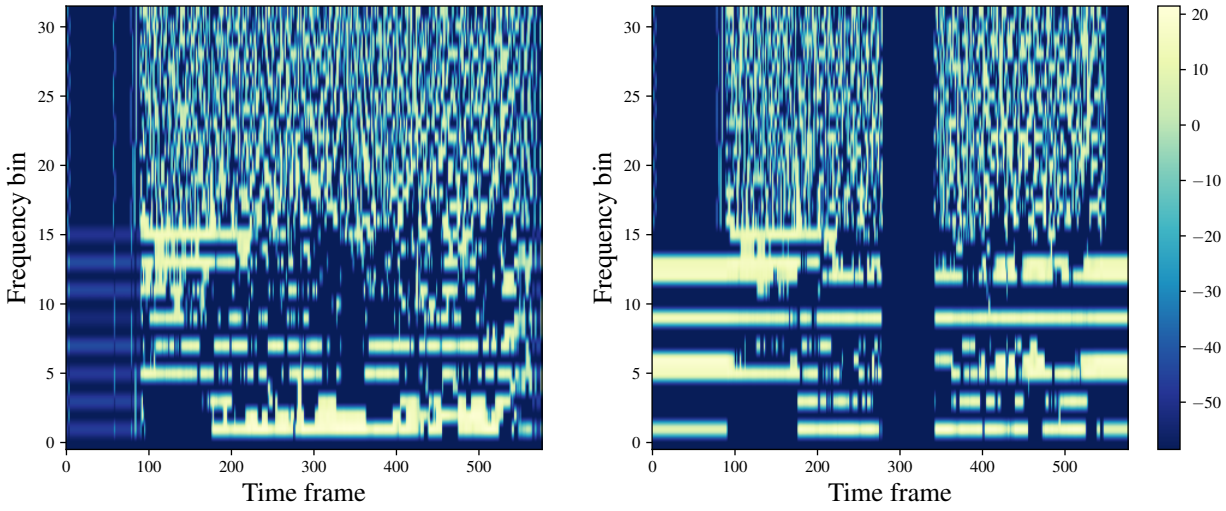
3.2.3.1 Spectral Augmentation

The first pre-processing method was completed by augmenting the audio features during training using the “SpecAugment” method. This method removes a small percentage of randomly selected data features. Some frequency and time feature vector components of the audio features are removed randomly, by making some values zero. This is to create a larger variety of training data and a more robust model. The “SpecAugment” approach is implemented after extracting the Mel spectrogram of the audio data and before the MFCC values, due to the MFCC values already being a smaller representation of the Mel spectrogram in terms of feature reduction. The Mel spectrogram has 81 features compared to 32 MFCC features. During each training batch, there is a 50% chance that the feature data will be augmented. If the data is randomly selected to be augmented there is a 50% chance that only one sequence of frequency domain values and one section of the time domain values will be removed. Otherwise, there will be two separate sequences removed in the frequency domain and two separate sequences removed in the time domain.

The frequency masking is completed by removing a random section of the Mel frequency domain. The size of the frequency bins to zero is randomly selected between 0 and a max frequency mask parameter of 15 bins out of the possible 81 Mel frequency bins. The time masking is completed by removing a random section of the time domain. The size of the time bins to zero is randomly selected between 0 and a max time mask parameter of 35 bins, where the time bins usually consist of over 100 bins depending on the audio data size. An example of the “SpecAugment” method can be seen in Figure 3.5(b) where different sections of the frequency bins and time frames are zeroed.

3.2.3.2 Time Stretching

The second pre-processing method was completed by applying time stretching of 10% randomly to the data during training, to ensure that the speed of the audio is considered. This method stretches or shrinks the audio data by 10% of randomly selected data features without modifying the pitch of the audio data. The time stretching approach is implemented after extracting the complex values of the original spectrogram of the audio data. During each training batch, there is a 50% chance that the



(a) Original MFCC and delta coefficients.

(b) Spectral augmented MFCC and delta coefficients.

Figure 3.5. A combined MFCC and delta coefficients example of an audio track sampled at 16kHz, with a female voice saying: "Maintain your health while you have it, it's easier". (a) Original version without 'SpecAug' implementation. (b) 'SpecAug' applied version where two sections of the time domain and two small sections of the frequency domain were removed.

feature data will be time stretched. If the data is randomly selected to be augmented there is a 50% chance that the data will be stretched by 10%, otherwise the data will be shrunk by 10%.

The time stretching method uses a phase vocoder to interpolate the information of the audio data by using the phase information from the spectrogram. The vocoder changes the phases of specific frequency components to stretch or shrink the sound. A time step matrix $\mathbf{T_s}$ is created, where s is the different time instances. The time step matrix has a new amount of time instances where each time instance contains a set of spectrogram frequency values. These time instances are between 0 and the new required time based on the stretching or shrinking. This creates the new size of the new time step matrix. Two new copies of the original spectrogram with the original time steps are created. The spectrogram values at time instance T_s of the original spectrogram are copied to the first copy $Spec_0$ and the spectrogram values at time instance $T_s + 1$ of the original spectrogram are copied to the second copy $Spec_1$. The phases and magnitudes of the copied spectrograms are used to create the final time stretched spectrogram. The new phase is calculated using multiple equations. The first equation

is:

$$P_{new1} = P_{Ang1} - P_{Ang0} - P_{Adv}, \tag{3.12}$$

where $PAng1$ is the angle of $Spec_1$, P_{Ang0} is the angle of $Spec_0$ and P_{Adv} is the phase advance. The phase advance is the expected phase change for each bin. P_{Adv} is a linear step vector, that steps from 0 to $\pi * l_{hop}$ with 201 equal steps inbetween, which is the number of frequency points and l_{hop} is the length between DFT windows. The phase is then kept between $-\pi$ and π by applying:

$$P_{new2} = P_{new1} - 2 \times \pi \times round\left(\frac{P_{new1}}{2 \times \pi}\right).$$

The expected phase change P_{Adv} is added back to the new phase again:

$$P_{new3} = P_{new2} + P_{Adv},$$

and lastly the new phase P_{new3} is concatenated to the phase of the original spectrogram and the cumulative summation of the phase vector is used for the new spectrogram.

The new phase can also be calculated by adding the phases P_{Ang1} and P_{Ang0} in the complex domain using $-\exp(-i(P_{Ang1} + P_{Ang0}))$, and then get the angle of the resultant complex number.

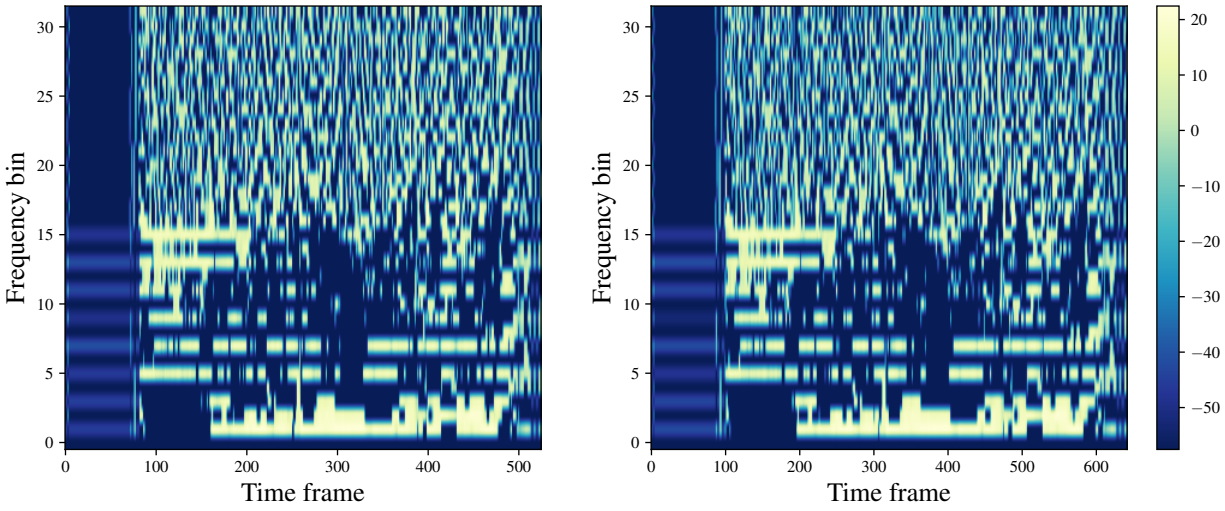
The magnitude of the new spectrogram is calculated as follows:

$$Mag_{new} = \alpha_{mod} \times norm_1 + (1 - \alpha_{mod}) \times norm_0, \tag{3.13}$$

where α_{mod} is the $T_s \% 1.0$, $norm_1$ is the magnitude of $Spec_1$ and $norm_0$ is the magnitude of $Spec_0$. Using the magnitude and phase of the new spectrogram, the final time stretched spectrogram is produced. In Figure 3.6(a), a time shrunk reproduction of the MFCC and delta coefficient features from Figure 3.4 is displayed. The spectral data remains very similar, but the time frames are decreased from 577 frames to 525 frames. In Figure 3.6(b), a time stretched reproduction of the MFCC and delta coefficient features from Figure 3.4 is displayed. The spectral data remains very similar, but the time frames are increased from 577 frames to 642 frames.

3.2.4 Text Data

The text data corresponding to the features and used as labels was pre-processed into lower-case, English alphabetic letters only. A few added classes were also included, such as a space and an apostrophe. These characters could change the way the audio is perceived and was therefore added. The text or label data was tokenized into numerical values, by simply mapping each character to an integer as seen in 3.1, for class vector creation to train the model. An extra class was added for



(a) Time shrunk MFCC and delta coefficients.

(b) Time stretched MFCC and delta coefficients.

Figure 3.6. A combined MFCC and delta coefficients example of an audio track sampled at 16kHz, with a female voice saying: "Maintain your health while you have it, it's easier". (a) Time shrunk implementation of 10% with 525 time frames. (b) Time stretched implementation of 10% with 642 frames.

unknown characters or letters not seen before by the model. The label data and the pre-processed audio data are not the same size and therefore each input value does not have a given class. To fix this for training, connectionist temporal classification (CTC) is used to align the audio data to the label data [16]. Each input is given an output distribution over all possible labels. An added pad class is added to the 29 classes for the CTC method to determine when the same character is being repeated. A pad token would be added in between repeating characters. In Table 3.1, the selected characters and their corresponding tokens can be inspected.

3.3 MODEL

The ASR model architecture, as in Figure 3.7, consists of a sequential CNN and transformer network that are connected with normalisation layers. The CNN expands the input audio features into multi-layered neurons or data points to be used by the transformer architecture. The time instances in a sequence of inputs are reduced by the CNN to enable more information to be stored as features using data over multiple time instances. The audio features are abstracted into a feature map consisting of 128 feature values at each time instance as input to the transformer architecture. The CNN consists

Table 3.1. A representation of the character-based tokenization, where each character class used in the ASR model is transformed to a number that is used for training the model.

Character	Token	Character	Token
<blank>	0	n	15
<space>	1	o	16
a	2	p	17
b	3	q	18
c	4	r	19
d	5	s	20
e	6	t	21
f	7	u	22
g	8	v	23
h	9	w	24
i	10	x	25
j	11	y	26
k	12	z	27
l	13	’	28
m	14	<unknown>	29

of a one dimensional convolution layer, a dropout layer and a normalisation layer. The output of the CNN is the input of a dense architecture, which changes the dimensions of the data and normalises the data for the transformer. The dense architecture consists of a fully connected linear layer and normalisation layer as well as a Gaussian error linear unit (GELU) layer and a dropout layer. The 4 layers are repeated twice, and each layer has a size of 128 feature inputs and outputs.

The transformer architecture has a two-headed layout, where each head contains two encoder layers and two decoder layers with a final feedforward layer that has a hidden layer size of 1024 nodes. A basic layout of one head of the transformer can be seen in Figure 3.7. The output of the transformer architecture is transferred to a final normalisation layer and dropout layer. The output of the dropout layer is reduced to 30 output classes using a final linear feedforward neural network (FFNN) layer.

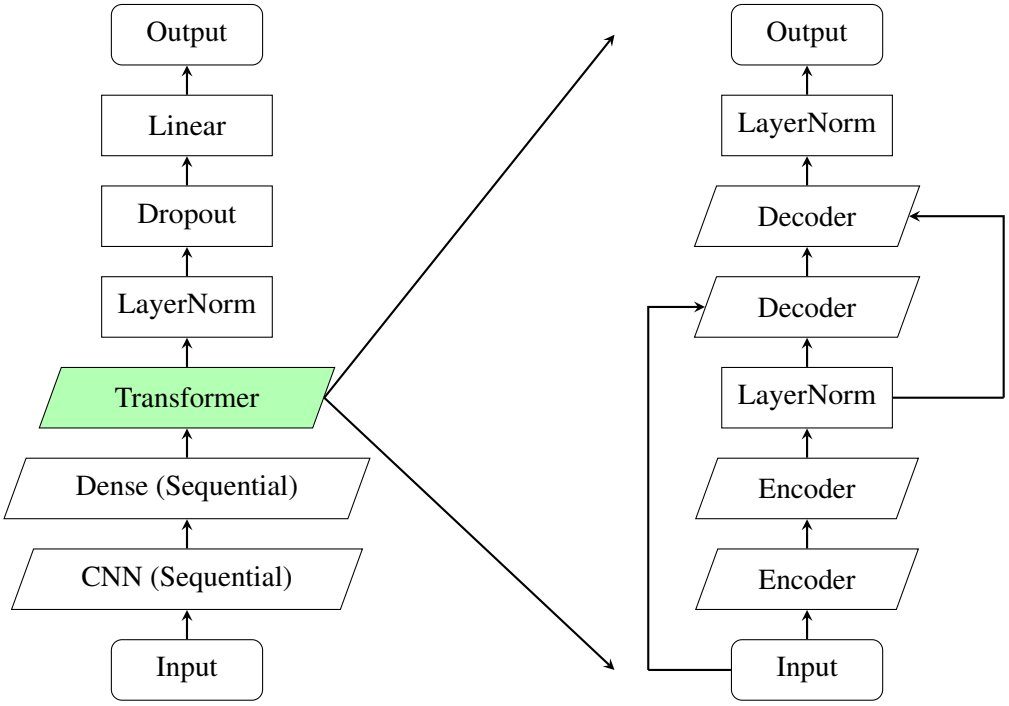


Figure 3.7. The flow diagram on the left, describing the basic layout of the architecture used to train the small-scale ASR system. The model consists of a one dimensional CNN layer and a dense layer to restructure and normalise the audio data. A transformer layer (right) is used to train the model according to the sequential correlation of the data. The normalisation and linear layers after the transformer are used to convert the data into 30 classes of the output characters of the model. The flow diagram on the right describes the two-headed transformer architecture. The diagram is a representation of one of the heads. Each head contains three encoder layers and three decoder layers with normalisation layers after the encoder and decoder.

3.3.1 Convolutional Neural Network Architecture

The convolutional neural network (CNN) architecture uses a convolutional operation to incorporate spatial context for the input data features in multiple dimensions. The MFCC and delta coefficients are represented by a two-dimensional matrix that represents the audio feature data in the frequency domain and the sequential time domain. The MFCC values and delta coefficients are normalised during pre-processing to reduce the bias of smaller and bigger values during training of the CNN by applying:

$$x'_{MFCC} = (x_{MFCC} - x_{MFCCmin})/(x_{MFCCmax} - x_{MFCCmin}). \tag{3.14}$$

The MFCC and delta input features act as channels in the CNN. The computational units used in a CNN layer are n-dimensional filters that are convolved with the input of each layer. The first layer input will

be the data, while the input of following layers will be the output of the previous layers. In this case, only one layer will be used for parameter reduction of the model. The filters used in the convolutional layers consist of randomly instantiated values that form a smaller dimensions than the input, therefore allowing spacial capturing of the input features in time, while reducing the dimensionality. The spatial capturing allows for a new relationship of the data features between the different input values. This allows for the sequential time-based features to be mapped during training. The same filter is used across the entire input sequence and allows for parameter reduction by sharing filter parameters between sequential inputs.

The CNN allows new local features to be learned based on the MFCC and delta coefficients, in each time step, used as inputs to the CNN architecture. New features such as noise changes based on different frequencies are learned. Stacking convolution layers can create more local features to be learned, but at a computational cost of using more parameters, memory, and training time. The main component of any CNN is the convolutional unit. The convolutional units are used as filters to extract new features from input data. A FFNN or linear layer produces a one-dimensional output layer, while a CNN generally produces multi-dimensional feature maps.

The filter in a convolutional layer consists of a grid of numbers or weights. The convolution between a section of the input matrix and the filter grid is applied, and the filter is moved across the input matrix horizontally based on a stride parameter l_{stride}. This process is repeated until the entire input matrix is convoluted with the filter matrix, essentially creating the output values for each channel of a CNN. The convoluted results are stored in an output matrix and the filter shifts horizontally with the stride length. The process is also known as cross-correlation which can be seen in Equation 3.15, but convolution has an added filter flip in the horizontal and vertical axis before multiplication and addition. This is not necessary in machine learning as the filter parameters or weights change during training of the model. Cross correlation is the measure of similarity or displacement between two different vector or functions. It is also known as the sliding dot product as a new matrix is formed by using the dot product on two matrices being cross-correlated. The cross correlation function can be described as:

$$\mathrm{R}_{\mathrm{XY}} \triangleq \mathrm{E}[\mathbf{XY}], \tag{3.15}$$

where X $= (X_1, ..., X_m)$ and Y $= (Y_1, ..., Y_n)$ are both vectors and this can be written component wise as:

$$\mathrm{R_{XY}} = \begin{bmatrix} \mathrm{E}[X_1Y_1] & \mathrm{E}[X_1Y_2] & \cdots & \mathrm{E}[X_1Y_n] \\ \mathrm{E}[X_2Y_1] & \mathrm{E}[X_2Y_2] & \cdots & \mathrm{E}[X_2Y_n] \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{E}[X_mY_1] & \mathrm{E}[X_mY_2] & \cdots & \mathrm{E}[X_mY_n] \end{bmatrix}.$$

The stride distance l_{stride}, affects the reduction in dimensionality between the input matrix and output matrix of a CNN layer. This is known as sub-sampling. The output matrix can be padded to keep the dimensionality constant, but a dimensionality reduction also results in a parameter reduction, which is one of the main goals of the model design. The output dimensions are given by:

$$l_{out} = \left(\frac{l_{in} - f_{conv} + l_{stride}}{l_{stride}} \right) \tag{3.16}$$

and

$$w_{out} = \left(\frac{w_{in} - f_{conv} + l_{stride}}{l_{stride}} \right), \tag{3.17}$$

where l_{out} x w_{out} represents the output dimension size and l_{in} x w_{in} represent the input dimension size. In our case l represents the time dimension w represents the Mel frequency dimension. The l x w image represents the entire spectrogram. The filter size is represented by f_{conv} x f_{conv} and l_{stride} is the stride length. In the case of audio processing, the 32 audio features are 32 different channels and cross-correlation from Equation 3.15 occurs for the sequential time units over all the channels. This requires only Equation 3.16 as the filter and the inputs would both have 32 channels. Adding slight padding does retain some time-based information and ensures that data is not lost between the changes for data over time. Adding padding changes Equation 3.16 to:

$$l_{out} = \left(\frac{l_{in} + 2 \times l_{pad} - f_{conv} - 2}{l_{stride}} + 1 \right), \tag{3.18}$$

where l_{pad} is the padding size.

3.3.1.1 Model Implementation

The CNN selected for this ASR model applies a small one-dimensional convolution over the time-based dimension input for all the 32 audio feature channels. The process can be described for each input sequence L with:

$$(N_{bat}, C_{out}) = b(C_{out}) + \sum_{k=0}^{C_{in}-1} w(C_{out}, k) \star in(N_{bat}, k), \tag{3.19}$$

where the input size is (N_{bat}, C_{in}, L) and N_{bat} is the batch size. The channels C_{in} and C_{out} are the number of input feature channels and output feature channels respectively and L is the sequential length of the input audio signal. The $\star$ symbol represents the cross-correlation operator. The CNN layer has a set of

bias values b and a set of weight values w that are adjusted during training using a loss function and back propagation to adjust the CNN to predict the correct classifications. The weight and bias values are randomly initialised using a random Gaussian distribution before training to ensure the model does not start training with a bias to a certain local minimum.

The model uses an input of multiple sequential time-based inputs l_{in} x w_{in}, where each time-based input has 32 channels or features. The output l_{out} is selected to have the same number of 32 channels, but half the time-based outputs. The convolving filter size f is selected to be 10 and a stride length l_{stride} of 2 is used. A zero padding value p of 5 was selected to add 5 zeros on each side of the input features. Therefore, using Equation 3.16, it can be determined that the output would be half the size after the convolution process occurs:

$$l_{out} = \left(\frac{l_{in} + 2 \times 5 - 10 - 2}{2} + 1\right),$$
$$\therefore \quad l_{out} = \left(\frac{l_{in}}{2}\right).$$

3.3.1.2 Model Additions

A normalisation layer is applied to the batch output data of the CNN to produce a normalised output y given x, such that:

$$y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \varepsilon}} \times \gamma + \beta, \tag{3.20}$$

where $E[x]$ is the mean of the probability distribution of the batch data and $\text{Var}[x]$ is the variance of the probability distribution of the batch data. The parameters, γ and β, are learnable parameters for each element in the data to ensure the data remains discriminate. A small value of 1×10^{-5} is added as ε to the variance of x, to ensure that division by zero does not occur if the variance is 0. The normalisation layer ensures that the feature values of the data are transformed to similar ranges and therefore reducing the risk of any vanishing or exploding gradient problems when an activation function is added during training.

The normalisation layer is followed by a Gaussian error linear unit (GELU) activation function, which is implemented to the data so that:

$$x = x \times \Phi(x), \tag{3.21}$$

and $\Phi(x)$ is the cumulative distribution function for a Gaussian distribution such that:

$$\Phi(x) = 0.5 \times x \times (1 + \text{Tanh}(\sqrt{(2/\pi)} \times (x + 0.044715 \times x^3))). \tag{3.22}$$

The GELU activation function introduces non-linearity to the data by adding weights to the inputs based on their percentile in the cumulative distribution function.

The final layer of the CNN system is a dropout layer. The dropout layer is used to randomly mask or zero selected channel features during training, using samples from a Bernoulli distribution to select which channel features are to be zeroed. Each channel has independent zero values on each forward training call. A probability p value determines the probability that an item is zeroed during training such that

$$z_i \sim \text{Bernoulli}(p), \quad z_i \in z,$$

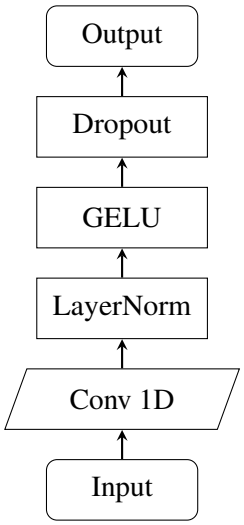
where the zero z_i is a Bernoulli distribution with a given probability p and z is the same size dimension as the input data. The randomly masked positions are then multiplied to the input data to zero the selected data. A dropout value of 0.1 was selected during training. The output values are scaled to $\frac{1}{1-p}$, to ensure that the data remains normalised. During evaluation the probability value is set to 0 and the module computes an identity matrix during the prediction forward call. A basic flow diagram of the CNN architecture is displayed in Figure 3.8(a) to understand the process of each of the layers in the CNN architecture.

3.3.2 Dense Architecture

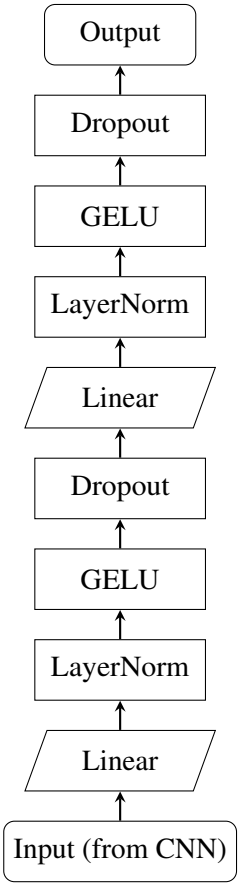
A dense architecture subsystem is appended to the output of the CNN subsystem. This subsystem consists of linear layers, normalisation layers, activation functions and dropout layers. The dense subsystem is used to normalise and reshape the data for the transformer architecture. The first layer is a linear layer or also known as a dense layer. This layer is used to change the dimensionality of the output from the input data given. In this case, the linear layers are used to change the dimensionality of the output of the CNN architecture and prepare the data for the main transformer architecture. The linear layer uses a weight matrix and matrix multiplication to transform input features into output features. The 32 input features from the CNN dropout layer are passed as a flattened one dimensional matrix and multiplied by a weight matrix. The output features are produced using a linear transformation:

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}, \tag{3.23}$$

where $\mathbf{y}$ are the output features, $\mathbf{x}$ are the input features, $\mathbf{W}$ represents the weight matrix and $\mathbf{b}$ represents the added bias parameters. The weight and bias values are randomly initialised, where the weight matrix $\mathbf{W}$ has a size of $(y_n \times x_n)$ where y_n is the number of output features and x_n is the number of input features. The bias vector $\mathbf{b}$ has a size of y_n. The values are initialised using the mean $\upsilon(-\sqrt{k}, \sqrt{k})$, where $k = \frac{1}{x_n}$. The first linear layer converts the 32 features x_n into 128 features y_n for each unit in



(a) A basic layout of the CNN architecture used in the small-scale ASR system.



(b) A representation of the dense architecture consisting of two sets of linear layers and additional normalisations, GELU and dropout layers.

Figure 3.8. (a) describes the basic layout of the CNN architecture used to in the small-scale ASR system. The CNN model consists of the main 1 dimensional convolutional layer and additional normalisation, GELU and dropout layers. (b) describes the dense architecture. The dense architecture consists of two sets of fully connected linear layers as the main layers. The architecture also has additional normalisations, GELU and dropout layers.

time.

The linear layer is followed by a normalisation layer identical to the normalisation layer used in the CNN architecture in 3.3.1.2. The normalisation layer has the same properties as in Equation 3.20, but only difference is that the amount of inputs is 128 features. An activation function layer is also introduced to allow the dense architecture to learn non-linear relationships. The GELU activation

function, as in Equation 3.21, is used with 128 features following the normalisation process. The final layer is another dropout layer to introduce masked elements into the model. A dropout probability p, of 0.1 was selected for the Bernoulli distribution probability.

The 4 layers in the dense architecture are repeated to ensure model stability during training. The 4 layers ensure that the parameters do not change drastically during training and therefore the received input data can be slightly different and still produce an accurate result. This increases prediction accuracy for the case of voice data where different people have different accents and produce different input features for the same output. The second linear layer has an identical number of input and output features of 128. The repeated normalisation layer, GELU activation function layer and dropout layer are identical to the first set of dense layers. The only changes are the weight and bias parameter values that are adjusted during training of the model. The dense architecture creates a linear relationship between the output features of the CNN and the input features of the transformer. The dense layer also allows for data feature normalisation before the data is received by the transformer architecture. A basic flow diagram of the dense architecture is displayed in Figure 3.8(b) to understand the process of each of the layers in the dense architecture.

3.3.3 Transformer Architecture

RNNs introduced the opportunity to train sequential data and by producing time-based predictive models. This was achieved by introducing feedback connections for sequential data processing. The transformer architecture is a modern evolution of neural architectures that replaced RNNs, such as LSTMs. Attention mechanisms were added to RNN networks for improved performance by reducing the vanishing gradient problem and adding tokens that depend on previous tokens. Attention mechanisms within RNN structures produces sequential tokens that stored information about the sequential data, but the processing had to be sequential. Removing the RNN structure and using purely attention-based mechanisms removed the need for sequential processing. Transformers consist of attention mechanisms on their own, without the RNN structure, eliminating sequential processing and using parallel processing [15]. Transformers make use of positional encodings of the sequential data to eliminate the remembering process of RNNs. A transformer consists of an encoder and decoder architecture as in Figure 3.7. The encoder layers have self-attention units that process the input data from previous encoders or the input sequence data and their relevant weight functions iteratively one layer at a time. The encoder layers produce output values that are transferred to a FFNN layer to process each output encoding individually for the next encoder as input. The decoder layers have

self-attention units as well as an additional attention mechanism that draws information from the outputs of the encoders. Each encoder and decoder layer are accompanied by a FFNN layer for residual connections and layer normalisation. The first decoder takes positional information and embeddings of the output sequence as input. Masking is used to block the decoder from inspecting current or future sequential output data, therefore preventing reverse information flow. The last decoder implements a final linear transformation and softmax layer to produce output probabilities for the given output classes or vocabulary [14]. The softmax activation function is a mathematical function that converts the output layer vectors of values into vectors of probabilities, where each vector of probabilities sums to unity. The probabilities of the values are proportional to the relative values stored in the vectors. The softmax function σ can be expressed as:

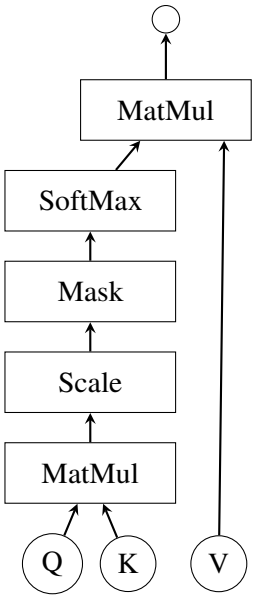
$$\sigma = \frac{e^{x_i}}{\sum_j^k e^{x_j}}, \tag{3.24}$$

where x_i represents a particular node in the vector and k represents all the values in the vector.

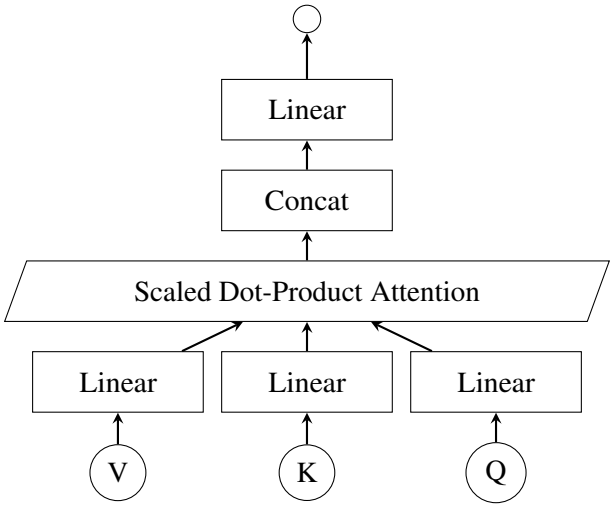
Transformer networks use scaled dot product attention units as the main mathematical functions in the encoders and decoders. Every attention unit input has a corresponding weight value. The attention weights are calculated between every token simultaneously to produce embeddings for every token in context. This context contains information about the token and a weighted combination from other relevant tokens, weighted by attenuation weights. Each attention unit consists of a query weight W_q, a key weight W_k and a value weight W_v matrix. Each token i consists of the input word embedding x_i, multiplied with each of the three weight matrices. This produces a query vector $q_i = x_i W_q$, a key vector $k_i = x_i W_k$ and a value vector $v_i = x_i W_v$. The attention weights are calculated from the query and key vectors, where each attention weight a_{ij} is the dot product of q_i and k_j from i to j. The attention weights are also divided by $\sqrt{d_k}$, which is the dimension of the key vector, to stabilise the gradient during training. Lastly, the attention weights are passed through a softmax function to normalise the data to sum to unity. The different weight matrices W_q and W_k allow for the model attention to be non-symmetric. If a token i attends to a token j, the dot product of the query vector q_i with the key vector k_j is large, but does not determine that token j will attend to token i. This is one of the reasons multiple weight matrices are used. The output of an attention unit for a token i is the weighted sum of the value vectors for all tokens weighted by a_{ij}. The attention calculation for all tokens is therefore a large matrix calculation for matrices Q, K and V representing all the tokens q_i, k_i and v_i. This produces the attention equation from [14],

$$Att(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{3.25}$$

One set of attention units with weights (W_q, W_k and W_v) is known as an attention head [14]. Each layer in a transformer model consists of multiple attention heads. One attention head attends to tokens that are relevant to each individual token. Multiple attention heads help the model to create different definitions of relevance between tokens. Multi-head attentions are processed in parallel and concatenated to be passed through a final FFNN layer for further processing. An illustration of the scaled-dot attention as in Equation 3.25, can be inspected in Figure 3.9(a) and an illustration of one layer of a multi-head attention can be inspected in Figure 3.9(b).



(a) Scaled dot-product attention.



(b) One layer of multi-head attention. Multi-head attention will stack attention layers and linear layers on top of each other to run in parallel, while the final concatenate layer and linear layer are used to combine the data from the multi-head attention layers.

Figure 3.9. (a) describes the basic layout of the scaled dot-product attention. (b) describes the multi head attention layers and how they run in parallel for multi-head attention.

3.3.3.1 Model Implementation

Multiple transformer architectures are used for the model experimentation, but the main transformer architecture consists of two heads and each head consists of two or three encoder layers and two or three decoder layers. The first encoder receives a $(t \times b \times x)$ dimensional input where t is the amount of time-based inputs, b is the batch size of 32 or 64 and x is the input features from the dense layer that consists of 128 features. The input and its positional encodings are used in the attention function from Equation 3.25 to produce an output of dimension $(t \times b \times x)$. The first encoder output is passed to the input of the second encoder and the process is continued for the amount of encoder layers to create

an encoder system. The first decoder receives a $(t \times b \times x)$ dimensional input of the input features from the dense layer as well as the output of the encoder system as a separate input. The input and its positional encodings are used in the attention function from Equation 3.25 to produce an output of dimension $(t \times b \times x)$ for the decoder. The current and future outputs are masked to ensure the model does not have reverse flow information. This would cause the model to have the value that is to be predicted already available. The following decoder layer uses the input of the previous decoder layer as well as the output of the encoder layer to calculate the output of the encoder layer. This process is continued for the amount of decoder layers to create a decoder system. An example of a two-encoder two-decoder layer transformer can be seen in Figure 3.7.

The encoder layer consists of multiple sub-layers. The first sub-layer is an attention layer that receivers the input of the encoder. A normalisation layer that receives both the input to the encoder and the output of the attention layer follows. The normalisation layer is followed by two linear layers. The first linear layer has 1024 nodes, and the second layer has 128 nodes to keep the output dimensions of the encoder the same as the input dimensions. The final layer of the encoder is a normalisation layer that receives the output of the final linear layer and the output of the first normalisation layer. The second encoder layer is identical to the first encoder layer but receives the output of the first encoder layer with dimensions $(t \times b \times x)$ as the input and has different weight values in the sub-layers. The second encoder layer produces an output with identical dimensions as it's input of $(t \times b \times x)$. The process is repeated for the given amount of encoders. The final layer in the encoder system is a normalisation layer to normalise the data for the decoder system. A representation of the encoder system is displayed in Figure 3.10.

The decoder layer consists of multiple sub-layers. The first sub-layer is an attention layer that receives the output of the dense layer. A normalisation layer that receives both the output of the dense layer and the output of the attention layer follows. The normalisation layer is followed by another attention layer that receives the output of the encoder system as well as the output of the first normalisation layer. The output of the second attention layer and the first normalisation layer is passed to a second normalisation layer. The second normalisation layer is followed by two linear layers. The first linear layer has 1024 nodes, and the second layer has 128 nodes to keep the output dimensions of the decoder the same as the input dimensions. The final layer of the decoder is a third normalisation layer that receives the output of the final linear layer and the output of the second normalisation layer. The second decoder layer is identical to the first decoder layer but receives the output of the first decoder layer and the

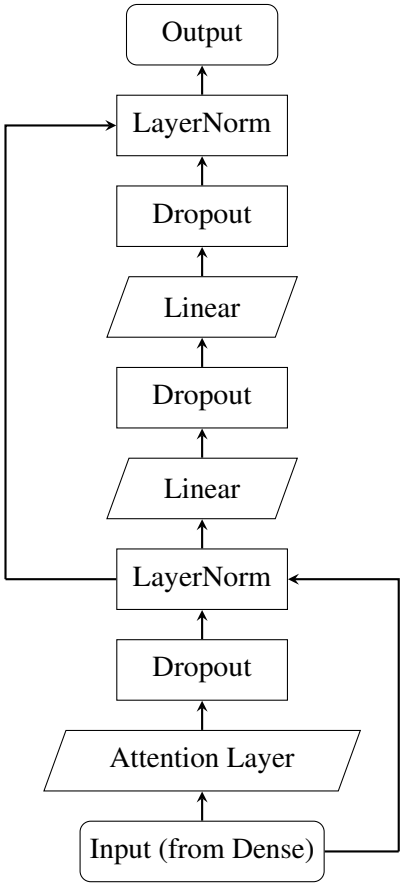


Figure 3.10. A representation of the encoder subsystem, consisting of an attention layer, two normalisation layers and two linear layers with 1024 nodes each.

output of the encoder system. The second decoder layer has different weight values in the sub-layers. The second encoder layer produces an output with identical dimensions as the input of $(t \times b \times x)$. The process is repeated for the given amount of decoders. The final layer in the decoder system is a normalisation layer to normalise the data for the final output of the transformer. A representation of the decoder system is displayed in Figure 3.11.

3.3.3.2 Model Additions

The transformer architecture is followed by a final normalisation, dropout, and linear layer. The normalisation layer is applied to the batch output data of the transformer to produce a normalised output as seen in Equation 3.20. The γ and β parameters from the equation are learnable parameters for discriminate data. The ε value remains 1×10^{-5} to remove the possibility of division by 0. The following dropout layer is used to randomly mask or zero selected elements during training, using samples from a Bernoulli distribution. Each channel has independent zeroes values on each forward

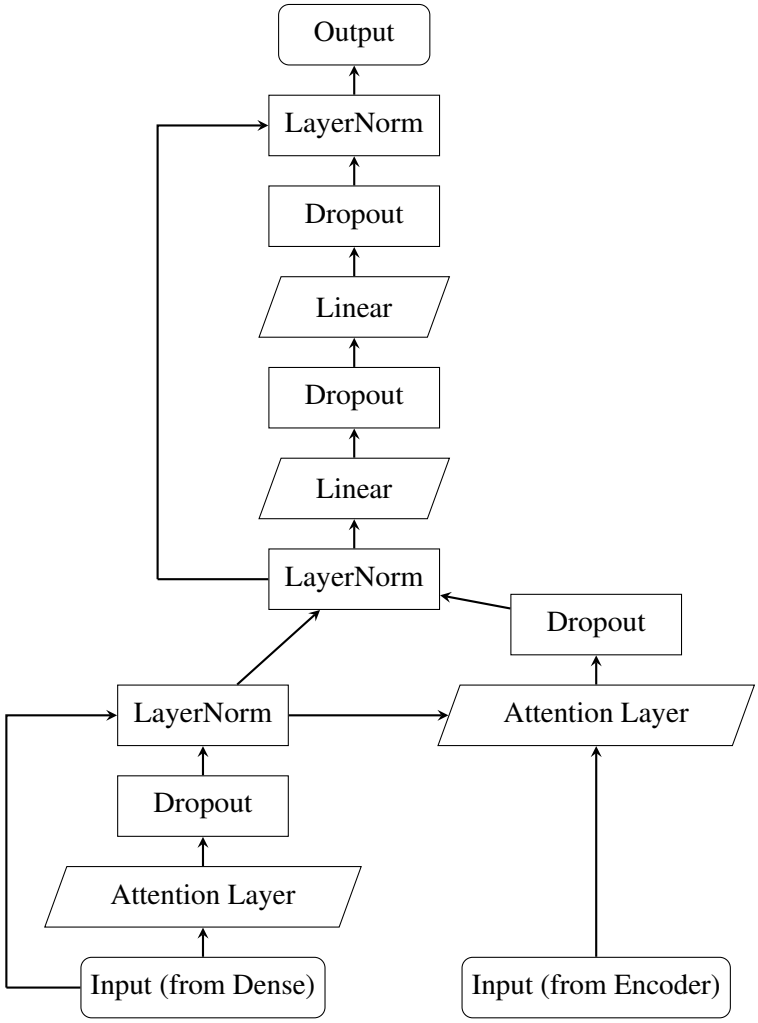


Figure 3.11. A representation of the decoder subsystem. The decoder receives two inputs. One from the previous encoder system, and another from the dense system before the encoders. The decoder has an attention layer for each of the inputs and two normalisation layers before the linear layers. There are two linear layers with 1024 nodes each and a final normalisation layer to produce the output data.

training call. A probability p value of 0.1 was selected to determine the probability that an item is zeroed during training. A final linear layer is added to the model to change the dimensions of the data from $(t \times b \times 128)$ to $(t \times b \times 30)$. The t represents the time-based dimension size and b represents the batch size. The final 30 features represent the 30 possible character classes of the model. A final softmax function layer is applied to enable a cost function to compare predicted output classes to the actual target classes during training.

3.3.4 Model Initialisation and Training

The weight and bias parameters are all randomly initialised before training. The weight and bias matrices define the different functions of the ASR model. The CNN weights and biases represent the convolution function, the dense weights and biases represent the linear function, and the transformer weights and biases represent the transformer function. All these weight and bias parameters map the ASR network and are adjusted and updated during the training process to achieve the desired label results by minimising a cost function E. Back propagation is used to update the weights and biases for all the subsystems in the ASR model by using gradient descent. Each weight associated with a node or data point in the model is updated using:

$$w_{ij}^{t+1} = w_{ij}^{t} - \eta \frac{\partial E}{\partial w_{ij}^{t}}, \tag{3.26}$$

where t is the current time step of the model and η is the learning rate. The learning rate must be small enough to avoid instabilities in the model by stepping over error local minima, but large enough to train the model effectively.

Cross-entropy loss is a cost function that could be used as it specialises in multi-class classification where outputs are one-hot encoded and there is only one correct class per input. The cross-entropy loss is calculated as:

$$L(\mathbf{p},\mathbf{y}) = -\sum_{no=1}^{N_{class}} y_{no} log(p_{no}), \tag{3.27}$$

where $\mathbf{p}$ is the predicted network output, $\mathbf{y}$ is the actual target vector with a one corresponding to the correct class and zeros as the incorrect classes. The value N_{class} is the total number of elements or classes in the vector. This loss function could not be used as each input feature set did not have a correlating character assigned to it.

The CTC loss function is a more specific loss function for ASR models and is selected for the model training. The other loss functions require each input to have a correlating output or target. CTC loss is used to calculate the loss between a continuous time series and a target sequence. Using the CTC algorithm, the maximum likelihood label is determined for each input value. A blank token is added to the character classes to ensure repeating characters remain repeating characters. The string of characters including the blank are the possible labels for each input. The maximum likelihood labels for each input value are compressed by removing all repeating characters without a blank token and finally removing the blank classes. This set of characters is the model output and is compared to the corresponding text label targets. The difference between the model output characters and text labels

targets is known as the CTC loss. The CTC loss is used to train the model and adjust the weights and biases of the model to minimise CTC loss. The mean difference between the predicted and actual labels is calculated as the CTC loss as in Equation 3.28. This loss function is used to adjust the weights and biases by using back propagation through the entire model. The probabilities of all the possible alignments of the input sequence and target sequence is summed to produce a loss value. In CTC loss, the label error rate (LER) can be defined as the normalised edit distance between the output classifications of a classifier model h_{ctc} and the classification targets such that:

$$LER(h_{ctc}, S') = \frac{1}{Y} \sum_{(x,y) \in S_{dat}} ED\left(h_{ctc}(x)\right), \tag{3.28}$$

where S_{dat} is a dataset with dimensions $(x \times y)$ and Y is the total number of target labels. The edit distance, $ED(p,q)$ is the distance between two sequences p and q which would be the predicted and target sequences respectively. The edit distance is the minimum number of insertions, deletions or substitutions required to change sequence p into sequence q [16].

The chain rule is used to calculate the gradients of the respective weights in the entire network for each layer L' and each node i using:

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}^{L'}} &= \frac{\partial E}{\partial p_i} \frac{\partial p_i}{\partial a_i} \frac{\partial a_i}{\partial w_{ij}^{L'}}, \\ &= \delta_i^{L'} x_j, \end{aligned} \tag{3.29}$$

and

$$\delta_i^{L'} = (p_i - y_i) h_i'(a_i),$$

where $\delta_i^{L'}$ is the error signal at node i for layer L'. The function $a_i = \sum_j w_{ij} x_j$ is the weighted sum of all the inputs to node i before an activation function and h_i' is the derivative of the mathematical function layer for each subsystem with respect to the function a_i. The predicted value p_i and the actual value y_i are the predicted and actual output values for node i and x_j is the output of the previous layer node connected with a weight w_{ij}^L. The gradient of the selected CTC loss cost function is calculated for each weight and each input. A batch of weights from multiple inputs are added together, and the weight updates are applied. The weight updates are iteratively applied during training to attempt to achieve a minimum loss function of zero.

The learning rate controls the size of the update step from gradient calculations. The model weights and biases are updated using Equation 3.26. The "AdamW" optimizer adds to this equation as not all the weights and biases require different learning rates for efficient learning of the model. The "AdamW"

optimizer uses the square of recent gradients and momentum to adjust different weights based on their gradient error. L2 regularisation is used to add the sum of squares of all the weights w_{all}, multiplied by a selected hyperparameter called the weight decay parameter wd, to the original loss ls_{init} value such that:

$$ls_{fin} = ls_{init} + wd \times \frac{\sum w_{all}^2}{2}. \tag{3.30}$$

This creates an updated weight decay equation that changes Equation 3.26 to:

$$w_{ij}^{t+1} = w_{ij}^{t} - \eta \frac{\partial E}{\partial w_{ij}^{t}} - \eta \times wd \times w_{ij}^{t}. \tag{3.31}$$

The weight decay equation is used in practice instead of the L2 regularisation as updating the cost function would add substantially more computations than adding $wd \times w_{ij}^{t}$ to the weight update function. The "AdamW" optimizer uses the weight decay equation with added momentum to produce:

$$w_{ij}^{t+1} = w_{ij}^{t} - \eta \times move_{avg} - \eta \times wd \times w_{ij}^{t},$$

where $move_{avg}$ is:

$${move_{avg}}^{t} = \alpha_{mom} \times {move_{avg}}^{t-1} + (1 - \alpha_{mom}) \times \frac{\partial E}{\partial w_{ij}^{t}}.$$

The α_{mom} coefficient is another hyperparameter to control the momentum. The "AdamW" optimizer only performs the weight decay equation after the parameter-wise step size is implemented. Therefore, the weight decay portion does not have moving averages and is proportional to each weight itself. For the selected "AdamW" optimizer an initial learning rate of 1×10^{-3} is selected. The weight decay parameter wd is selected to be 1×10^{-2}. Both these values are selected based on general starting parameter values in machine learning [62].

The learning rate controls the size of the update step from gradient calculations. If the learning rate is too large, the model might never converge, and if the learning rate is too small, the model might get stuck in local minima. Learning rate scheduling is added to find the perfect balance for the learning rate value. Generally, the learning rate starts at a relatively large value for an initial learning rate which is still very small, to ensure the model does not become stationary in a local minimum during initial training. As the training of the model progresses, the learning rate is decreased to ensure the model converges during training. A "reduce on plateau" scheduler is added that reduces the learning rate by a factor of 0.5, if no loss value improvement has been recorded for 6 iterations. Different factors values are tested for optimum training efficiency. The goal is to ensure the model trains with as few iterations as possible and still reaching the lowest possible loss value.

3.4 LANGUAGE MODELS

Language models are added to the end-to-end character-based ASR model to enhance the understanding of words from the predicted characters. The character-based ASR model learns different sounds and characters and can develop a correlation between the given character classes and spaces. A language model improves the ASR model by using the predicted characters and changing them into existing words. A language model adds natural language processing (NLP) to the existing ASR model and adds word understanding. There are two types of language models namely deterministic and statistical models. Deterministic language models are defined by grammar rules and are not as common in NLP. Statistical language models use the probability of occurrence of a sequence of words based on previous words. These are the language models typically used in NLP [23]. Two popular statistical language models presented in literature are the n-gram language model and the BERT transformer-based language model [34, 23]. The n-gram language models are smaller and practical for a small-scaled ASR system but lack the diverse correlation between words and deep sequential understanding of sequences of words that the BERT language model contain.

3.4.1 N-gram language model

N-gram language models were one of the first language models used in NLP [13]. The model uses a statistical probability of words occurring based on previous and following words. Statistical language modelling estimates the prior probability $P(W)$ for a sequence or string of words W in a total vocabulary V. The vocabulary V can be thousands to hundreds of thousands of words depending on the selected language model. The sequence of words W is usually broken into small sequences or sentences that are conditionally independent so that:

$$P(W) = \prod_{i=1}^{n} P(w_i|w_1, w_2, ..., w_{i-1}). \tag{3.32}$$

The surrounding words $W_{k-1} = (w_1, w_2, ..., w_{i-1})$ are used in a function $\phi(W_{k-1})$ to determine the predicted word w_k. Therefore, Equation 3.32 can be rewritten as:

$$P(W) \simeq \prod_{i=1}^{n} P(w_k|\phi(W_{k-1})).$$

Language models use different methods to find a function ϕ to estimate $P(w_k|\phi(W_{k-1}))$. The n-gram language model uses $\phi(W_{k-1}) = w_{k-n+1}, w_{k-n+2}, ...w_{k-1}$, so that the 'n+1' number of words before the selected word is taken as $\phi(W_{k-1})$.

The basis of an n-gram model uses count statistics. The probability that the word 'happy' occurs after 'I am', is the number of times 'I am happy' occurs divided by the number of times 'I am' occurs so

that:

$$P(\text{'happy'}|\text{'I am'}) = \frac{C(\text{'I am happy'})}{C(\text{'I am'})},$$

where C is the number of times the sequence of words or the word appears in the vocabulary. Therefore:

$$P\left(W_n|W_1^{n-1}\right) = \frac{C\left(W_1^{n-1}W_n\right)}{C\left(W_1^{n-1}\right)}. \tag{3.33}$$

The n-gram language model uses the basic idea mentioned above and expands it for sequencing probabilities, for a collection of words, by using a combination of Bayes theorem and Markov assumptions. Bayes theorem states that:

$$P(B|A) = \frac{P(A,B)}{P(A)}, \tag{3.34}$$

and

$$P(A,B) = P(A)\cdot P(B|A).$$

This rule can be expanded for multiple occurrences in sequence such that there are more than two elements that are correlated. This can represent multiple words in a sequence so that:

$$P(A,B,C,D) = P(A)\cdot P(B|A)\cdot P(C|A,B)\cdot P(D|A,B,C).$$

Equation 3.33 and Equation 3.34, is in essence how the probabilities of the words are calculated for every sequence using an n-gram language model. Certain words of the sequence might not be in the vocabulary or corpus of the language model, but these words can be excluded while still determining the probability of the word given the rest of the words in the sequence without the non-vocabulary words. Using the Markov assumption Bayes equation can be changed to only use the last 'n' words given the n-gram model. For a trigram model, Equation 3.34 is changed to:

$$P(A,B,C,D) = P(A)\cdot P(B|A)\cdot P(C|A,B)\cdot P(D|B,C),$$

and the n-gram model probability can be redefined as:

$$P\left(W_n|W_1^{n-1}\right) \simeq P\left(W_n|W_{n-N+1}^{n-1}\right). \tag{3.35}$$

To measure the performance of an n-gram language model, different quality measures are used. Perplexity and WER are generally used for language models [10]. The log perplexity of a model $PP(W)$, can be calculated as:

$$PP(W) = -\frac{1}{m}\sum_{k=1}^{m} log_n\left[P(W_i|W_{i-1})\right], \tag{3.36}$$

for an n-gram language model. The word error rate as calculated in Equation 1.2 is a better quality measure as the final result is word prediction from a combination of an end-to-end ASR model and an n-gram language model.

3.4.1.1 N-Gram implementation

Once the end-to-end ASR model has predicted the output text, the text is inserted into a 4-gram language model. Different 4-gram language models were tested and created using the kenLM toolkit from [34]. The toolkit uses Equation 3.33 to create probabilities for word relationships as in Equation 3.35. The toolkit created a large dictionary with relationships between all the words in the corpus for 1-gram to the selected n-gram relationship. A 4-gram model was selected as any larger model would be too big for a small-scale ASR system. The 4-gram model data was collected from the text corpus of the 'LibriSpeech data' with 200 003 different word entries. In Table 3.2, the amount of combinations for each n-gram with a probability attached to each combination is provided. Other pre-existing 4-gram language models are also tested and compared from different text corpuses.

Table 3.2. 4-gram language model with word combinations for 1-gram to 4-gram relationships. Model created using the LibriSpeech dataset corpus.

N-gram	Number of combinations
1-gram	200 003
2-gram	38 229 161
3-gram	45 941 329
4-gram	60 975 692

Each output sequence of the ASR model, consisting of the predicted classes based on the speech input, is transferred to the 4-gram language model. A beam search method is implemented where the output of the entire sequence with the best probability is selected over a beam search value of 500 different output sequences. Each beam search output sequence is transferred to the 4-gram language model, word for word. For each sequence, the language model recommends a word based on the previous three words, until the sequence is complete. If the predicted word has a high probability, the original output word in the sequence might be changed. The language model receives two weighting ratio values namely alpha α_{lang} and beta β_{lang}. The α_{lang} value gives a weighting associated with the language model probabilities. An α_{lang} value of 0 would mean the language model has no effect and a value of 1 would take the total probability of the language model prediction. The β_{lang} value is the weight associated with the probabilities of different words in the beam search. Multiple combinations of α_{lang} and β_{lang} values were tested to find the ideal values of $\alpha_{lang} = 0.4$ and $\beta_{lang} = 0.85$. An alpha value that is too high changes the words to satisfy the language model probabilities, but the original words

from the speech are more susceptible to being replaced. This is not desired in a speech recognition model. The final output of the language model is converted from tokens to characters and joined to produce the predicted output sequence.

3.4.2 BERT language model

The drawback of using n-gram language models, is that only a few words, prior the predicted word are used in the prediction process. RNNs can use the entire sequence to predict a word, without using excessive amounts of computing memory. RNNs use the same number of parameters for word sequences with different lengths. Transformer-based models do the exact same as RNNs, but are even more memory and parameter use efficient. The bidirectional encoder representations from transformers (BERT) model uses data in the entire sequence to predict or correct words in a sequence. Other transformer-based models such as generative pre-trained transformer (GPT) 3 models and transformer 5 (T5) models are too large for speech correction. They are used for summarization and text creation. The BERT model uses only the encoder part of a transformer model and encodes the entire sequence with positional embeddings to predict the most probable word for each given word in a sequence. The BERT model uses the data from the sequence in both directions of the masked word that must be predicted [23].

The BERT base model consists of 12 layers or transformer encoder blocks, with 12 attention heads and over 110 million parameters. The BERT large model has 24 layers or transformer encoder blocks with 16 attention heads. The BERT large model has just over 340 million parameters. The model is pre-trained with a large collection of English words from the 'BookCorpus' with 800 million words and English Wikipedia with 2.5 billion words. During training, a random selection of 15% of the input data is masked at 80% of the time. The other 10% of the time, the randomly selected 15% is replaced with a random token (word or character) and the last 10% of the time, the randomly selected 15% is kept as is. The BERT language model uses a masked language modelling approach, where one word in the sequence is masked and predicted based on all the surrounding words in the sequence.

Each word in the sequence is tokenized and positional embedding or encodings are added to the word to determine the position of each word in the sequence. The positional information is crucial for the model to understand the sequential order of words. Selected words are then masked and predicted. The positional encodings PE, have the same dimensions d as the model and can be defined as sine and

cosine functions where:

$$PE_{(pos,2i)} = sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right), \tag{3.37}$$

and

$$PE_{(pos,2i+1)} = cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \tag{3.38}$$

where pos is the position and i is the dimension. This allows each dimension of the positional encoding to correspond to a sinusoid. The wavelengths are selected from 2π to $10000 \cdot 2\pi$ and allows the model to attend to relative positions with a fixed offset. Different embedding values can be represented as a linear function of PE. It's important to note that the positional encodings play a critical role in helping the model understand the sequential context of words within the input sequence. By incorporating positional information, the model can capture the relationships between words based on their positions in the sequence, which is essential for tasks such as language modeling and speech recognition.

The model assumes that the words surrounding the masked word in the sequence are correct and therefore if a large amount of words in a sequence are incorrect the language modelling will not be as accurate. The tokenized words and positional embeddings are delivered to a set of encoder layers that are similar to the encoder in Figure 3.10 from the transformer architecture in the speech recognition model. Using many parallel transformer heads with multiple encoder layers and attention mechanisms as in Equation 3.25, the model learns different relationships between words in a sequence in terms of positioning in the sequence and word correlations. The BERT model uses transfer learning where the base BERT model is fine-tuned for different tasks by changing the parameters of the final layer of the BERT model for the required task such as sentence correction in our case for speech recognition.

3.4.2.1 BERT implementation

The BERT model was fine-tuned using a 'Huggingface' library in python to ensure that the vocabulary of the model matches the vocabulary of the data used in the end-to-end speech recognition model. The fine-tuned model continued to have a larger vocabulary than the data used in the end-to-end speech recognition model, to ensure that the model could predict other words from the English dictionary that do not occur in the Common Voice dataset and LibriSpeech dataset. This creates a more generalised model.

The BERT model is implemented with the output data of the beam search method, where the predicted sequence with the highest probability is used. The highest predicted output sequence is received by the BERT language model. Each word in the sequence is iteratively masked, and the language model predicts the 50 most probable words for the masked word. The language model predicted words are

compared to the original word before masking and the word most similar to the original word is used. This method ensures that the language model does not attempt to change words in the sequence that were not present in the original speech. The output words of the language model are also checked to ensure that only tokens from the original character classes of the ASR model is used. If a word is changed or corrected using the language model, the input sequence is updated before the next word in the sequence is masked and predicted by the BERT language model.

3.5 CHAPTER DISCUSSION

In Chapter 3, the methodology, design, and initial experiments for the small-scale, transformer-based ASR model were presented. In Section 3.2, various data pre-processing techniques were explored, including audio data sampling, feature extraction methods like the Mel-Spectrogram and MFCC, and text tokenization. In Section 3.3 the model's design involved a modular approach, allowing for flexibility in adjusting size and hyperparameters, utilizing both CNN and transformer architectures. In Section 3.4 different language models, including traditional N-gram and BERT models, were integrated into the ASR framework. Overall, Chapter 3 provides a comprehensive overview of the development process, setting the stage for the subsequent experimentation and analysis in the following chapters.

CHAPTER 4 EXPERIMENTS

4.1 CHAPTER OVERVIEW

This section describes the different experiments and training methods used to train the designed ASR model using the different datasets as well as different pre-processing techniques and evaluation techniques. Section 4.2 provides the first attempt at training the ASR model and compares the model with different parameters and compares the model to a similar model with an LSTM architecture instead of a transformer architecture. In Section 4.3, the different pre-processing techniques and output methods are tested for the Common Voice dataset. Section 4.4 describes the experiments in training the ASR model with the ideal pre-processing techniques and output methods for the Common Voice dataset and LibriSpeech dataset.

4.2 MODEL COMPARISONS

4.2.1 Batch Size and Pre-Processing

The first designed model was selected to be a small transformer-based model with a single attention head and a single encoder/decoder layer. The model has 32 input features with 16 MFCC values and 16 MFCC delta values described in Section 3.2.2. The data is a subset of the Mozilla Common Voice English 7.0 dataset [18] and is sampled at 8kHz. The model has 30 classes describing each character in the alphabet and other important symbols for ASR as in Table 3.1. The model has an initial learning rate of 1×10^{-3} and a dropout value of 0.1 for all the dropout layers during training. The model weights and biases are randomly initialised. The evaluation subset of the Common Voice English dataset is used to evaluate the model after every epoch and adjust the learning rate if necessary, using the “AdamW" optimizer. If the model training is stuck at a local minimum for CTC loss over a few training iterations, the optimizer increases the learning rate to change the weight and bias training values more rapidly and to improve training efficiency. Training is continued until the model reaches a minimum CTC loss value for the evaluation data and the loss value does not decrease over 10 iterations.

This prevents overtraining the model for the specific training data that is provided. The first model is tested with different batch sizes. A batch size of 16, 32 and 64 produced a CTC loss value of 2.9758, 2.4326 and 2.038 after 150 epochs, respectively.

The data pre-processing was changed to produce 81 Mel spectrogram features instead of the 32 MFCC and delta features for the same model with a batch size of 64 and produced a CTC loss of 2.606 after 150 epochs. In Table 4.1 it is indicated that the reduced MFCC features produced significantly better results than the 81 Mel spectrogram features for a batch size of 64 input sequences at 150 epochs.

Table 4.1. The first experiment testing the CTC loss of a 1-headed transformer, with 1 encoder layer and 1 decoder layer using a subset of the Common Voice training and validation dataset. The different model experiments have different batch sizes and features.

Model (1 head transformer)	**Features**	**Batch Size**	**Sampling Rate (kHz)**	**LR**	**Epochs**	**CTC loss (train)**	**CTC loss (valid)**
Experiment 1	MFCC + Delta (32)	16	8000	1×10^{-3}	150	2.9061	2.9758
Experiment 2	MFCC + Delta (32)	32	8000	1×10^{-3}	150	2.3867	2.4326
Experiment 3	MFCC + Delta (32)	64	8000	1×10^{-3}	150	**2.2558**	**2.0380**
Experiment 4	Mel Spectrogram (81)	64	8000	1×10^{-3}	150	2.6131	2.6068

4.2.2 Architecture Comparison

The transformer-based model in Section 4.2.1 with 32 features and a batch size of 64 is compared to an LSTM architecture with a single LSTM layer. The single LSTM layer model contains 4 790 141 parameters, while the single head transformer only contains 762 109 parameters. The LSTM architecture produced a CTC loss of 2.0265 while the transformer architecture produced a similar CTC loss value of 2.038 with 4 million fewer parameters.

The transformer-based model in Section 4.2.1 was increased to a two-headed transformer architecture, with two encoder layers and two decoder layers to produce a CTC loss value of 1.412 after 150 epochs.

In Table 4.2, the comparison between parameters and CTC loss results for the two different transformer architectures and the LSTM architecture is presented.

Table 4.2. Different architectures are tested to determine which architecture produced the best CTC loss value for a subset of the Common Voice dataset and taking the architecture parameter size into account.

Model	Parameters	Features	Batch Size	Sampling Rate (kHz)	LR	Epochs	CTC loss (train)	CTC loss (valid)
LSTM (1 layer)	4 790 141	MFCC + Delta (32)	64	8000	1×10^{-3}	150	1.7713	2.0265
Transformer (1 head : 1 enc + 1 dec)	762 109	MFCC + Delta (32)	64	8000	1×10^{-3}	150	2.2558	2.0380
Transformer (2 head : 2 enc + 2 dec)	1 488 125	MFCC + Delta (32)	64	8000	1×10^{-3}	150	**1.7393**	**1.412**

Table 4.3 displays the model parameter sizes for different architectures. All the architectures use the same batch size of 64 with 32 input data features. Due to experimenting on smaller models only and setting a VRAM limitation of 12 GB, larger models were not tested as more VRAM was required. The transformer architecture increases parameters based on the amount of encoder layers and decoder layers. The number of heads does not affect the trainable parameters, but they do affect the GPU VRAM due to parallel processing. The two layer LSTM architecture and transformer architecture with more than two heads is too large to train with the available VRAM. The two-headed transformer, with more than two encoder layers and two decoder layers is also too large to train when using a batch size of 64 for the training data.

The two-headed transformer model with a batch size of 64 was selected to train the large Common Voice dataset. In Figure 4.1, the average CTC loss of a subset of the Common Voice training data is provided per epoch. The model was trained using the MFCC and delta audio features, sampled at 8kHz. The results indicate that the ASR model trained to minimise the CTC loss value until a minimum value was reached. The training was stopped to prevent overtraining. In Figure 4.2, the average CTC loss for a subset of the Common Voice validation data is provided per epoch. The validation data is used to optimise the training of the model. When the validation CTC loss value did not drop for 5 epochs,

Table 4.3. Different architectures and their parameter counts.

Model	Parameters
LSTM (1 layer)	4 790 141
LSTM (2 layers)	13 186 941
Transformer (1 head : 1 enc + 1 dec)	762 109
Transformer (2 head : 2 enc + 2 dec)	1 488 125
Transformer (2 head : 3 enc + 3 dec)	2 214 141
Transformer (2 head : 4 enc + 4 dec)	2 940 157

the “Adam” optimizer changed the learning rate of the model to allow a larger jump in weight values.

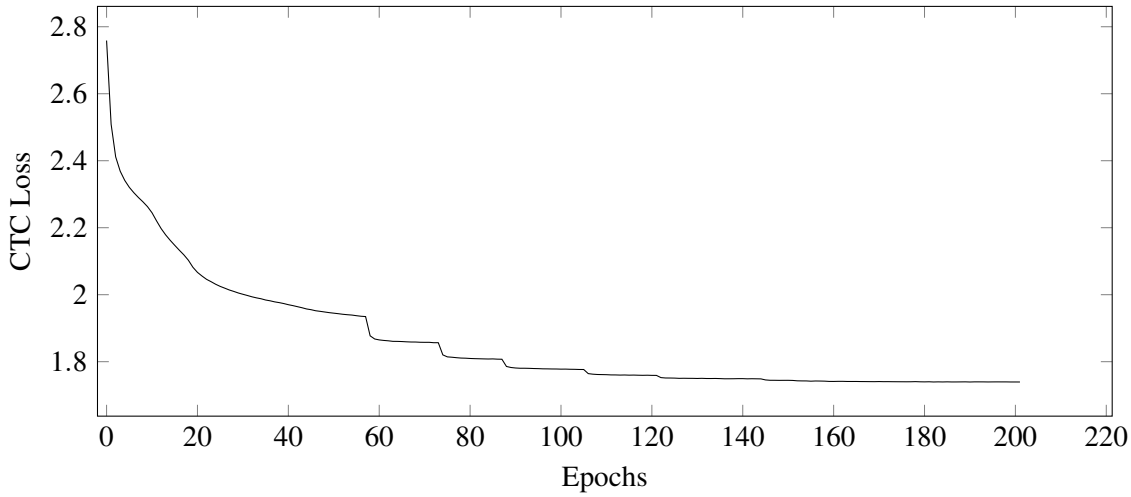


Figure 4.1. Average training CTC loss per epoch for the small-scale two head transformer model, using a subset of the Common Voice 7.0 English training dataset. The graph indicates that the average CTC loss value decreased from 2.78 to 1.73.

4.3 PRE-PROCESSING AND OUTPUT METHOD

The end-to-end transformer-based ASR model with a two head transformer architecture is fully trained using the entire Common Voice 7.0 English training dataset and validation dataset. The trained model is evaluated using a subset of 500 sequences from the Common Voice testing data. The evaluation metrics used are the CER and WER so that the model can be evaluated as an actual ASR system. CTC loss represents a loss value that the model attempts to minimise during training.

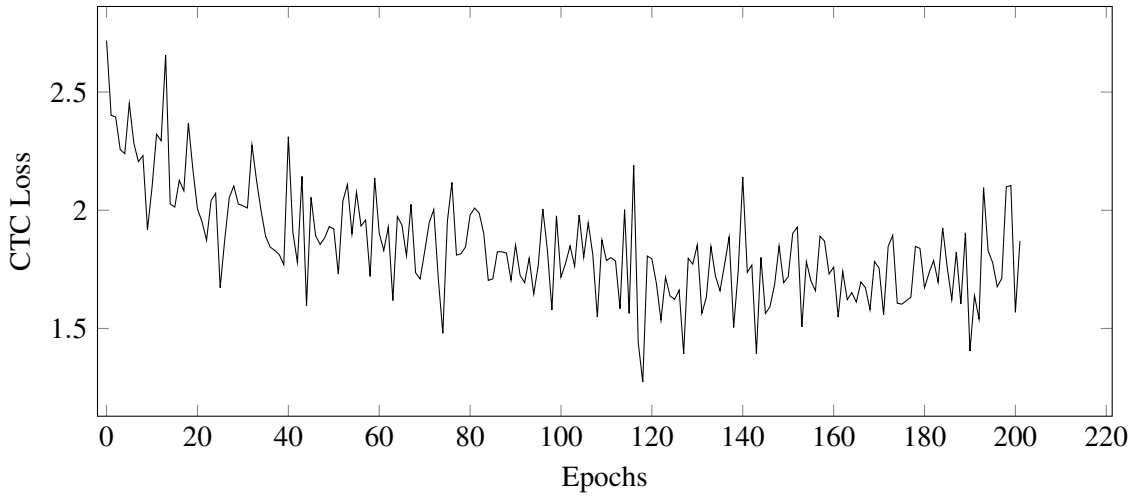


Figure 4.2. Average Validation CTC loss per Epoch for the small-scale two head transformer model, using a subset of the Common Voice 7.0 English training dataset. The validation data was used to optimise the model by changing the learning rate of the model based on the output of the average validation CTC loss per epoch.

4.3.1 Output method and language models

The output of the ASR model is used to predict the text from the given audio speech. Different methods are used to predict the output text as seen in Table 4.4. Each original model is tested using different output methods. The greedy decoding method takes the highest probability output character for each input audio segment and combines the characters using the CTC compression to produce output characters. The beam search method used the most likely sequence of characters based on the top 500 output sequences. A spell checker toolkit called 'Textblob' added as an output method to determine if a normal spell checker would improve the ASR model. The final output method was a language model. The 4-gram language model was selected as it is a large dictionary of words with a statistical probability that they are close to each other. A larger transformer-based language model, BERT is also tested, but it consists of a model with billions of parameters and defeats the purpose of building a small-scale ASR model. The CERs and WERs are not very low, but this is due to the small testing sample selected. The BERT language model produces worse results than the smaller n-gram language model, and therefore it was decided that the 4-gram language model would be used with the ASR model. The 4-gram language model is also smaller than the BERT language model and therefore reduces the system complexity with respect to the number of parameters.

Table 4.4. CERs and WERs for different output methods of the Common Voice 7.0 validation data.

Output Method	**Performance Metrics**	
Performance Metrics	**CER %**	**WER %**
Greedy Decoding	43.00	90.43
Beam Search 500	41.99	91.04
4-gram lm (LibriSpeech)	41.91	86.43
4-gram lm (CV + LibriSpeech)	41.82	74.27
BERT	42.84	89.56
Textblob Spell Checker	42.92	87.64

4.3.2 Common Voice Pre-processing

In Table 4.5, the different CERs and WERs are displayed for different pre-processing techniques and different model output methods. A higher sampling rate was selected to determine if 16kHz would improve the results when compared to 8kHz. The testing data used was a small segment of the Common Voice validation dataset consisting of 500 audio segments. In Table 4.5, the beam search method with an added 4-gram language model produces the best results for all data pre-processing types.

Table 4.5. CERs and WERs for different sampling frequencies and pre-processing methods of the Common Voice 7.0 validation data.

Pre-Processing Performance	**Greedy Decoding**		**Beam Search 500**		**Beam Search 500 and 4-gram lm**	
Performance Metrics	CER %	WER %	CER %	WER %	CER %	WER %
Normal MFCC 8kHz	43.00	90.43	41.99	91.04	41.82	74.27
MFCC with Spec Augment 16kHz	41.31	86.79	39.62	86.91	36.59	67.40
Normal MFCC 16kHz	40.63	86.66	38.45	85.50	35.15	65.22
MFCC with time stretching 16kHz	35.84	82.01	34.17	80.08	29.98	59.83

The first model was created by using the MFCC and delta audio features during training. This model was trained with a sampling rate of 8kHz and 16kHz respectively. The results show that the audio data features sampled at 16kHz produced a CER of 35.15% and a WER of 65.22% compared to a CER of 41.82% and a WER of 74.27% using the 8kHz sampled audio features. The "SpecAugment" method increased the CER and WER slightly. This is most likely due to the audio features being MFCC values and their deltas and not a full spectrogram, therefore, the "SpecAugment" method removes too much necessary data from the audio features. Time stretching was introduced instead of the "SpecAugment" method. The time stretching method randomly stretched or shrank the length of audio segments by 10% while keeping all the audio data constant. The time stretching improved the CER to 29.98% and the WER to 59.83% as seen in Table 4.5.

4.4 COMMON VOICE AND LIBRISPEECH

The experiments in Section 4.2 and Section 4.3 determined that the ideal ASR architecture to use is the two-headed transformer with two encoder layers and two decoder layers. The ideal sampling frequency of the audio data is 16kHz and the ideal batch size is 64. The time-stretching pre-processing method during training improves the model accuracy as the time-stretching creates more data from the original dataset. The best output method to predict text from the audio is determined to be a beam search method with a 4-gram language model.

The Common Voice dataset and LibriSpeech dataset are separately trained and tested using the designed end-to-end ASR model. A final model is trained using a combination of the Common Voice dataset and LibriSpeech dataset. The Common Voice model, LibriSpeech model and combination model are compared to existing modern ASR models.

Another end-to-end ASR model with a transformer architecture consisting of two heads, but three encoder layers and three decoder layers is also created and trained using both the LibriSpeech and Common Voice dataset. The larger ASR model can run on the limiting 12 GB VRAM if the batch size of the data is reduced to 32. All other aspects of the model are identical to the smaller ASR model. The larger end-to-end ASR model is compared to the smaller ASR model as well as existing modern ASR models.

4.5 CHAPTER DISCUSSION

Chapter 4 provided a detailed overview of the experiments conducted to train the ASR model, utilizing various datasets, pre-processing techniques, and evaluation methods. In Section 4.2, we explored

the initial training attempts of the ASR model, comparing different parameters and architectures. Specifically, experiments were conducted to compare the performance of the model with varying batch sizes and pre-processing techniques, as well as to contrast it with an LSTM architecture. Section 4.3 focused on testing different pre-processing techniques and output methods for the Common Voice dataset. Experiments were carried out to evaluate the impact of output methods and language models on the model's performance, along with exploring specific pre-processing techniques tailored for the Common Voice dataset. In Section 4.4, experiments involving the training of the ASR model with optimal pre-processing techniques and output methods for both the Common Voice and LibriSpeech datasets were discussed. This section provided insights into the effectiveness of various techniques when applied to different datasets, highlighting the importance of dataset-specific approaches in ASR model training.

Overall, Chapter 4 encapsulated the experimentation phase of our ASR model development, showcasing the iterative process of refining parameters, exploring pre-processing techniques, and evaluating performance across diverse datasets. These experiments laid the groundwork for the subsequent results presented in the following chapters.

CHAPTER 5 RESULTS

5.1 CHAPTER OVERVIEW

This section describes the different results captured during the testing of the two end-to-end transformer-based ASR models designed and selected in Section 3 and Section 4. The ASR models are tested on the Common Voice and LibriSpeech datasets. Section 5.2 provides the training results for the two-headed transformer architecture with two encoder layers and two decoder layers as well as the training results of the two-headed transformer architecture with three encoder layers and three decoder layers on the Common Voice training and validation dataset. In Section 5.3, the two models are tested using the Common Voice testing dataset and compared to pre-existing ASR models. Section 5.4 describes the results for the two models tested on the LibriSpeech testing dataset.

5.2 TRAINING

In Figure 5.1, the average CTC loss of the Common Voice training data is provided per epoch. The model was trained using the MFCC and delta MFCC audio features, sampled at 16kHz, and time stretching random audio segments during training. The main architecture used for the training was a two-headed transformer with two encoder layers and two decoder layers. The results indicate that the ASR model trained to minimise the CTC loss value until a minimum value was reached. The training was stopped to prevent overtraining. In Figure 5.2, the average CTC loss for the Common Voice validation data is provided per epoch. The validation data is used to optimise the training of the model. When the validation CTC loss value did not drop for 5 epochs, the “Adam” optimizer changed the learning rate of the model to allow a larger jump in weight values. In Figure 5.1, there are sudden drops in CTC loss at a few epochs. This is due to the optimizer changing the learning rate of the model.

In Figure 5.3, the average CTC loss of the Common Voice training data is provided per epoch. The

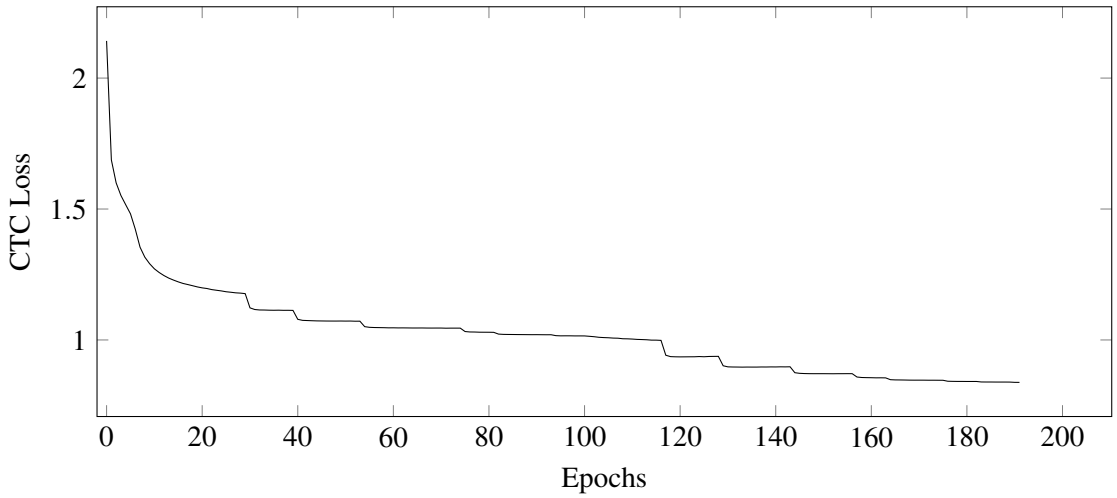


Figure 5.1. Average training CTC loss per epoch for the smaller transformer model, using the Common Voice 7.0 English training dataset. The graph indicates that the average CTC loss value decreased from 2.1419 to 0.8380. The audio data was sampled at 16kHz and converted to 16 MFCC values and 16 MFCC delta values. Time stretching of 10% was applied to the training data to simulate more data. 30 possible character classes were selected as possible outputs of the model.

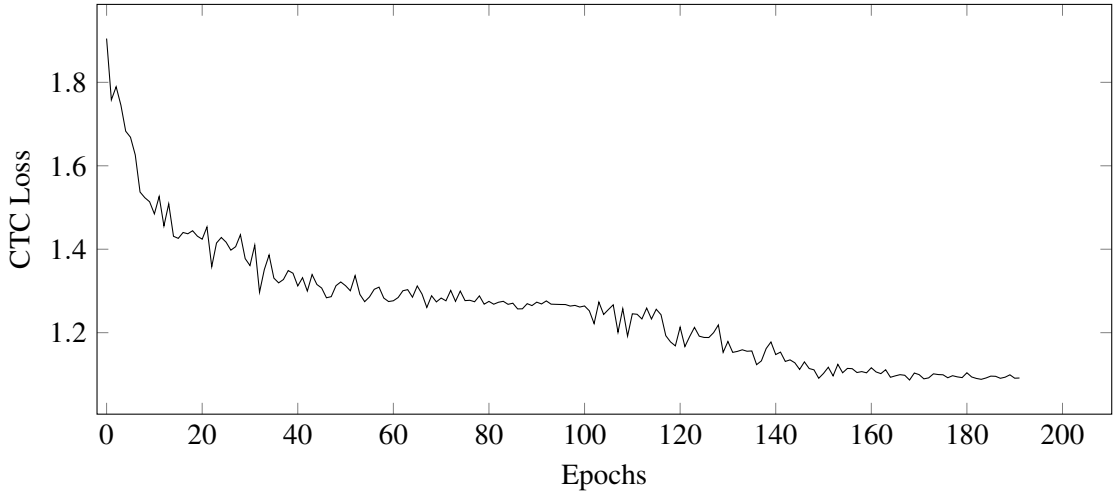


Figure 5.2. Average validation CTC loss per epoch for the smaller transformer model, using the Common Voice 7.0 English validation dataset. The graph indicates that the average CTC loss value decreased from 1.9049 to 1.0914. The validation data was used to optimise the model by changing the learning rate of the model based on the output of the average validation CTC loss per epoch.

model was trained using the MFCC and delta MFCC audio features, sampled at 16kHz, and time stretching random audio segments during training. The main architecture used for the training was a two-headed transformer with three encoder layers and three decoder layers. The results indicate that the ASR model trained to minimise the CTC loss value until a minimum value was reached. The training was stopped to prevent overtraining. In Figure 5.2, the average CTC loss for the Common Voice validation data is provided per epoch. The validation data is used to optimise the training of the model using the "AdamW" optimizer. In Figure 5.3, there are sudden drops in CTC loss at a few epochs. This is due to the optimizer changing the learning rate of the model.

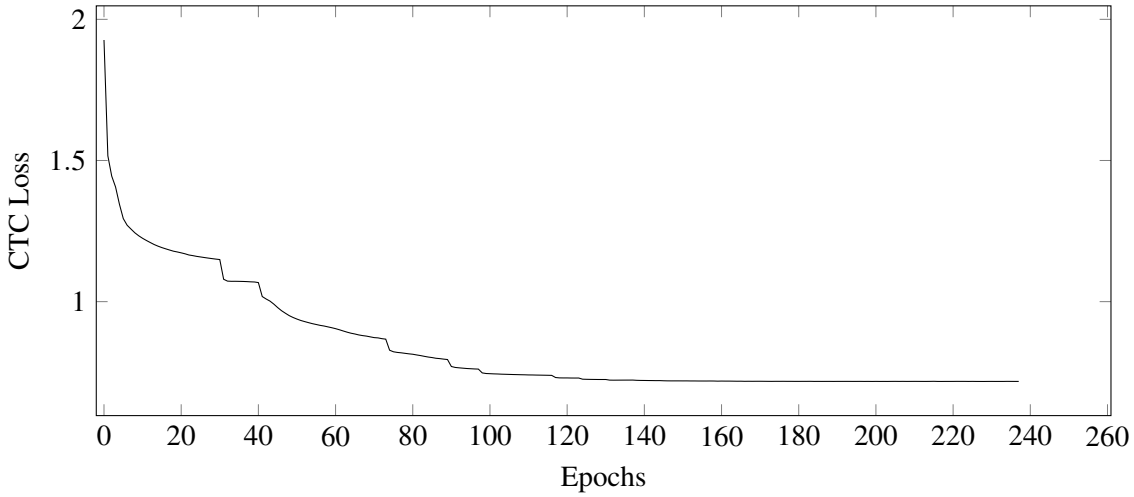


Figure 5.3. Average training CTC loss per epoch for the larger transformer model, using the Common Voice 7.0 English training dataset. The graph indicates that the average CTC loss value decreased from 1.9268 to 0.7174. The audio data was sampled at 16kHz and converted to 16 MFCC values and 16 MFCC delta values. Time stretching of 10% was applied to the training data to simulate more data. 30 possible character classes were selected as possible outputs of the model.

The training results indicate that the ASR model with three encoder layers and three decoder layers performs better than the ASR model with two encoder layers and two decoder layers when training and evaluating a model on the Common Voice 7.0 English dataset. The two layered architecture produced a final CTC evaluation loss of 1.0914, while the three layered architecture produced a final CTC evaluation loss of 0.9818. This is an improvement of 0.1096 for a parameter increase of 50%. Both models are tested using the Common Voice testing dataset to determine the CER and WER improvement.

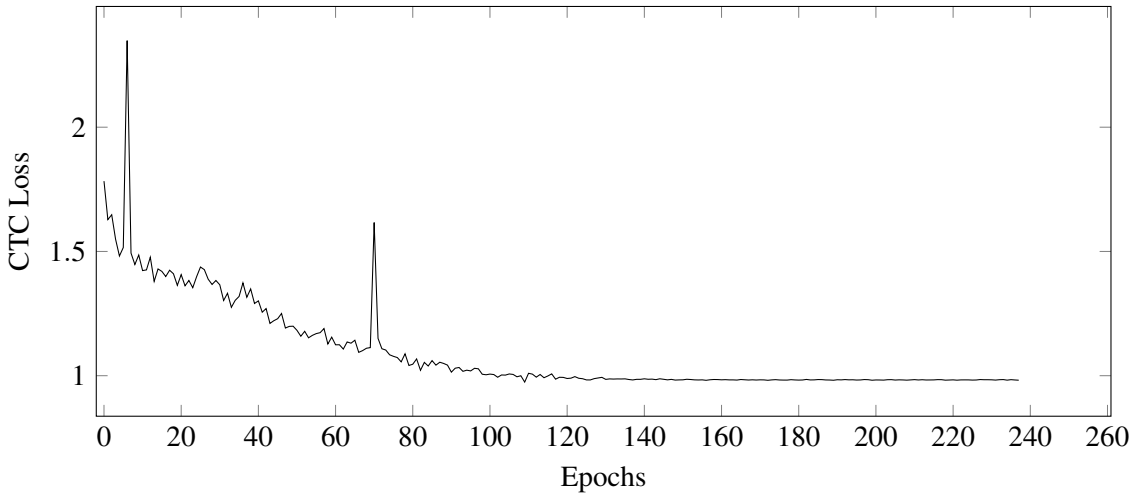


Figure 5.4. Average validation CTC loss per epoch for the larger transformer model, using the Common Voice 7.0 English validation dataset. The graph indicates that the average CTC loss value decreased from 1.7829 to 0.9818. The validation data was used to optimise the model based on the output of the average validation CTC loss per epoch.

5.3 COMMON VOICE RESULTS

Table 5.1 compares the two final trained small-scale transformer models results to existing large transformer-based models on the Common Voice validation dataset and testing dataset. With reference to Table 5.1, it is interesting to note that the small-scale transformer ASR models do not perform as well as the larger Wav2Vec2 models from [60] and [58]. The ASR models from previous work consist of architectures that have over 300 million parameters, while the models presented in this research only consists of 1.49 million parameters and 2.21 million parameters respectively. This is a parameter reduction by a factor of approximately 150. The other models are also pre-trained with over 50 000 hours of speech data, before being fine-tuned with the Common Voice dataset. The small-scale transformer-based models are only trained on the 2000 hours of Common Voice data.

The end-to-end small-scale transformer with two encoder layers and two decoder layers and an added 4-gram language model produced a CER of 24.87% and a WER of 52.51% on the validation dataset and a CER of 29.98% and a WER of 58.57% on the testing dataset. The end-to-end small-scale transformer with three encoder layers and three decoder layers and an added 4-gram language model produced a CER of 22.54% and a WER of 48.17% on the validation dataset and a CER of 27.89% and a WER of 54.50% on the testing dataset. The larger transformer-based architecture ASR model

Table 5.1. Comparison of the small-scale transformer-based ASR models presented in this research, to other large transformer-based ASR models described in literature, using the Common Voice validation dataset and testing dataset.

Common Voice Testing			**Validation Data**		**Testing Data**	
	Parameters	Hours Trained	CER %	WER %	CER %	WER %
End-to-end small-scale transformer (2head: 2 enc, 2dec)	1 488 254	2000 CV	24.87	52.51	29.98	58.57
End-to-end small-scale transformer (2head: 3 enc, 3dec)	2 214 141	2000 CV	22.54	48.17	27.89	54.50
Wav2Vec2 XLS-R 300 [60]	300 million	436000 Pre-Train + 2000 CV	7.36	30.71	N/A	N/A
Wav2Vec2 XLSR53 [58, 59]	317 million	56000 Pre-Train + 2000 CV	7.69	19.06	11.65	27.72
Wav2Vec2 XLSR53 With LM [58, 59]	317 million	56000 Pre-Train + 2000 CV	6.84	14.81	11.01	20.85

produces slightly better results, therefore, the parameter count does influence the accuracy of the ASR model.

The random chance of a single character being guessed correctly by the model is 3.33% and a sequence of characters reduces that probability exponentially. Therefore, a CER of 29.98% and 27.89% for both the models respectively is a good result for small-scale speech recognition models. The CER and WER could be improved by using a larger language model or training the model with more audio data. From the results it is observed that a small improvement in CER allows the language model to better understand words and produces a better WER. The CER difference between the smaller ASR model and the larger ASR model for the Common Voice testing data is 2.09%, but the difference in WER is larger at 4.07%.

The CERs from the larger existing transformer-based ASR models are less than half of the small-scale ASR models. However, it is important to note that the comparison is limited due to the larger existing

models being trained on 50 000 hours more speech data, so the parameter count is not the only influential factor between the models from this research and pre-existing models.

5.4 LIBRISPEECH RESULTS

5.4.1 LibriSpeech Testing

Table 5.2 and Table 5.3 compares the results of the two small-scale transformer models trained on only 2000 hours of Common Voice data and the same models trained on 1000 hours of LibriSpeech data. The models are also trained on a combination of the Common Voice dataset and LibriSpeech dataset. The models are tested on the LibriSpeech dev and test datasets. The fine-tuned model, with the Common Voice training dataset and the additional training data from LibriSpeech, outperforms the original Common Voice trained model and the original LibriSpeech trained model drastically. The larger transformer-based architecture with three encoder layers and three decoder layers performs better than the smaller transformer-based architecture with two encoder layers and two decoder layers.

Table 5.2. LibriSpeech Dev and Test results for a small-scale transformer-based ASR model, with two encoder layers and two decoder layers, trained on the Common Voice dataset and the same model being fine-tuned with the LibriSpeech training dataset.

LibriSpeech Testing	**Dev Clean**		**Dev Other**		**Test Clean**		**Test Other**	
End-to-end small-scale transformer (2 head: 2 enc, 2 dec)	CER %	WER %	CER %	WER %	CER %	WER %	CER %	WER %
Trained on Common Voice	15.42	34.81	28.49	55.82	15.74	35.31	29.59	57.40
Trained on LibriSpeech	10.61	25.49	23.44	47.93	10.61	25.51	24.48	49.82
Trained on Common Voice and LibriSpeech	8.25	20.55	19.95	41.73	8.52	20.76	21.07	43.92

In Table 5.2 the CER of the smaller ASR model, using the test-clean dataset, is reduced from 15.74% to 8.52% and the WER is reduced from 35.31% to 20.76%. The overall results for the model on the LibriSpeech data is a lot better than the Common Voice data, as the LibriSpeech data consists of more clear speech and is only in an American accent. The model also performs better with the added LibriSpeech training data due to an additional 1000 hours of audio data used to train the model. The LibriSpeech training data is also more similar to the LibriSpeech dev and test data.

Table 5.3. LibriSpeech Dev and Test results for a small-scale transformer-based ASR model, with three encoder layers and three decoder layers, trained on the Common Voice dataset and the same model being fine-tuned with the LibriSpeech training dataset.

LibriSpeech Testing	**Dev Clean**		**Dev Other**		**Test Clean**		**Test Other**	
End-to-end small-scale transformer (2 head: 3 enc, 3 dec)	CER %	WER %	CER %	WER %	CER %	WER %	CER %	WER %
Trained on Common Voice	13.57	30.61	26.29	51.39	13.89	30.93	27.49	53.58
Trained on LibriSpeech	12.08	28.91	25.54	51.85	12.55	30.14	26.56	53.75
Trained on Common Voice and LibriSpeech	6.04	15.45	15.90	33.47	6.40	16.03	16.73	35.51

In Table 5.3 the CER of the larger ASR model, using the test-clean dataset, is reduced from 13.89% to 6.40% and the WER is reduced from 30.93% to 16.03%. The overall results for the model on the LibriSpeech data is significantly better than the Common Voice data, as the LibriSpeech data consists of more clear speech and is only in an American accent. The model also performs better with the added LibriSpeech training data due to an additional 1000 hours of audio data used to train the model. In addition, the LibriSpeech training data is more similar to the LibriSpeech dev and test data when compared to the Common Voice training data.

The larger ASR model CERs and WERs in Table 5.3 performed better overall in comparison to the smaller ASR model CERs and WERs in Table 5.2 for the models trained on both the Common Voice and LibriSpeech training datasets. The difference in CER and WER for the LibriSpeech test clean dataset was 2.12% and 5.31% respectively. The larger ASR model has 2 214 141 parameters, while the smaller model has 1 488 254 parameters, which is a difference of 725 887 parameters. The larger model is therefore 48.77% larger than the smaller model. The 48.77% increase in model size improved the WER for the test clean data by 5.31%, indicating that an increase in parameters or layers in a transformer architecture has a significant impact on model performance, but only if sufficient training data is available for training the model.

The larger ASR model WER and CER in Table 5.3 performed worse, when using less training data, than the smaller ASR model CER and WER in Table 5.2 for the models trained on only the 1000 hours

of LibriSpeech training dataset. The difference in CER and WER for the LibriSpeech test clean dataset was 1.94% and 4.63% respectively. The smaller ASR model produces better results for the LibriSpeech dataset, even though the larger ASR model has more layers and parameters. This indicates that an increase in the number of parameters or layers in a transformer architecture has a negligible effect on model accuracy if insufficient data is available for training the ASR model. The larger model would theoretically perform better as it can create better paths between the input speech data and output speech text, but due to the data being limited to only 1000 hours, the smaller model performed slightly better than the larger model.

5.4.2 LibriSpeech Comparisons

Table 5.4 presents the WERS of the two final small-scale transformer models, trained on Common Voice and LibriSpeech training data. The two models' WERs can be compared to existing large transformer-based models' WERs using the LibriSpeech dev dataset and test dataset. With reference to Table 5.4, it is interesting to note that the small-scale transformer ASR model does not perform as well as the larger Wav2Vec2 and large transformer models but is comparable to older GRU-based models when it comes to WERs of the LibriSpeech dev dataset and test dataset. The transformer-based ASR models from previous work consist of architectures that are 150 times larger than the models in this research, based on the parameter count. The other models are also pre-trained with over 53 000 hours of speech data, before being fine-tuned with the LibriSpeech dataset. The small-scale transformer is only trained on the 3000 hours of Common Voice and LibriSpeech data.

The larger end-to-end small-scale ASR model, with an added 4-gram language model produced a WER of 16.03% and 35.51% on the test-clean and test-other data respectively on the LibriSpeech dataset. The model also produced a WER of 15.45% and 33.47% on the dev-clean and dev-other data respectively. The small-scaled end-to-end ASR model with an added 4-gram language model produced the given CER and WER using a model that was trained and tested using less than 12GB of video RAM, which was the limitation for this model. The other large transformer models require large clusters of graphics cards to train models that require over 120GB of video RAM. The result from an older GRU-based model in [25], produced a WER of 31.1% for the test other dataset, which is relatively close to the WER achieved by the best small-scale transformer, but at a significantly larger computational cost, as the architecture had 52.5 million parameters, which is a factor of 24 times larger than the architecture from this research. The small-scale transformer trains computationally faster, as it optimises parallel processing with the use of transformer architecture layers. The larger modern

Table 5.4. LibriSpeech dataset WER results for two small-scale transformer models compared to very large ASR models using mainly transformer-based architectures. The small-scale transformer models have 1.49 million parameters and 2.21 million parameters respectively, with an added 4-gram language model. All the other large scale transformer models contain over 52.5 million parameters without their added language models.

LibriSpeech Testing			**Dev Clean**	**Dev Other**	**Test Clean**	**Test Other**
	Parameters	Hours Trained	WER %	WER %	WER %	WER %
End-to-end small-scale transformer (2 head: 2 enc, 2 dec)	1 488 254 + lang model	1000 LibriSpeech	25.49	47.93	25.51	49.82
End-to-end small-scale transformer (2 head: 2 enc, 2 dec)	1 488 254 + lang model	2000 Pre-Train + 1000 LibriSpeech	20.55	41.73	20.76	43.92
End-to-end small-scale transformer (2 head: 3 enc, 3 dec)	2 214 141 + lang model	1000 LibriSpeech	28.91	51.85	30.14	53.75
End-to-end small-scale transformer (2 head: 3 enc, 3 dec)	2 214 141 + lang model	2000 Pre-Train + 1000 LibriSpeech	15.45	33.47	16.03	35.51
CTC-based GRU model [25]	52.5 million	Pre-Train + 1000 LibriSpeech	N/A	N/A	11.9	31.1
CTC-based GRU model with 4gram LM [25]	52.5 million + lang mod	Pre-Train + 1000 LibriSpeech	N/A	N/A	8.3	24.4
CTC-Based Transformer [54]	322 million	53800 Pre-Train + 1000 LibriSpeech	2.99	7.31	3.09	7.40
CTC-Based Transformer with transformer LM [54]	322 million + lang mod	53800 Pre-Train + 1000 LibriSpeech	2.63	6.20	2.86	6.72
Wav2Vec2 Transformer [55]	300 million + lang mod	53200 Pre-Train + 1000 LibriSpeech	3.4	6.0	3.8	6.5
Wav2Vec2.0 Transformer [20]	317 million + transformer lang mod	53200 Pre-Train + 1000 LibriSpeech	1.6	3.0	1.8	3.3
Crowd-sourced human level performance [57]	N/A	N/A	N/A	N/A	5.83	12.69

transformer-based ASR models outperformed with WERs of under 5% for the test clean dataset and WERs of under 10% for the test other datasets. This is due to the models being trained with about 50 000 hours more speech data. The models also have over 300 million parameters with multiple deep layers to create different mathematical paths, with different weights, between the input data and the

output classes.

All the examples selected were CTC-based ASR models that use character-based recognition with added language models to form the words for a given language. These models are the state-of-the-art ASR models, as they require the least amount of classes and parameters to train a very accurate ASR model. Older word-based models have up to 80 000 classes, based on the amount of possible words that can be predicted.

5.5 CHAPTER DISCUSSION

Chapter 5 presented the results of testing two end-to-end transformer-based ASR models on both the Common Voice and LibriSpeech datasets. In Section 5.2, we discussed the training outcomes of these models on the Common Voice dataset, showcasing variations in performance based on the number of encoder and decoder layers. Section 5.3 provided an analysis of the models' performance on the Common Voice testing dataset, highlighting their competitive performance compared to pre-existing ASR models. In Section 5.4, we detailed the results of testing the models on the LibriSpeech dataset. Notably, training the models with a combination of Common Voice and LibriSpeech data resulted in significant improvements in performance.

Overall, the smaller-scale transformer ASR models demonstrated promising performance, showcasing competitive results compared to larger models while utilizing reduced computational resources. These findings underscore the potential of small-scale ASR models for practical applications, paving the way for further exploration and refinement in the field.

CHAPTER 6 DISCUSSION OF RESULTS

6.1 RESEARCH OBJECTIVES

The main objective of the research was to determine if it is possible to build a small-scale speech recognition system that is comparable in CER and WER to larger modern ASR systems. The hypothesis was that a small transformer-based ASR model will produce similar WER results to traditional ASR models and compete with modern ASR systems that are exponentially larger in computational size. The results in Chapter 5, provided evidence that a small-scale end-to-end ASR model is comparable to older GRU-based ASR models and modern transformer-based ASR models when it comes to WERs, but modern transformer-based models produced a better WER accuracy due to the fact that they all were significantly larger in parameter size than the models designed in this research. The modern ASR models had over 300 million parameters while the models in this research had 1.48 million parameters and 2.21 million parameters respectively. That is a factor difference of roughly 150. The modern ASR models were also pre-trained on at least 53 000 hours of data before being fine-tuned for the LibriSpeech dataset that was used to measure the model performances. The models used in this research were pre-trained on 2000 hours of Common Voice training data before being fine-tuned for the LibriSpeech dataset that was used to measure the model performances. Therefore, the modern models were trained with 50 000 more hours. With all the given limitations, the final end-to-end ASR model still produced accurate results when compared to the larger modern ASR models and human level performance.

The results from the experiments provide sufficient evidence that a small-scale ASR system is a viable approach for under resourced languages with only small datasets available. The models can be pre-trained on different language data that uses the same alphabet or characters, and then be fine-tuned for the given language. The experiments also provide evidence that the quantity of training data plays a role in the model performance. In Figure 5.3 a larger model produces similar results to a smaller

model due to the amount of training data being insufficient. Larger datasets make the models more generalised for different speakers and accents or dialects. The following research questions were answered in the research:

- Question: Is it possible and viable to reasonably combine an acoustic model and a language model into one speech recognition model that will successfully convert speech to text?
- Answer: Most modern ASR models use an end-to-end approach that combines the acoustic model characteristics and language model characteristics into a single model. This results in a model directly converting audio data into text, without the phoneme calculation step in between. Adding a small secondary language model improves ASR model performance even more when it comes to WERs as the language model assists in combining characters into understandable words and words into understandable sentences.
- Question: What will the effect, of combining an acoustic and language model, be to the word error rate of an automated speech recognition system?
- Answer: End-to-end ASR models produce better word error rates, and the models are exponentially smaller. Language models are still very beneficial to ASR model accuracies.
- Question: How does a transformer model compare to other existing models when applied to speech recognition processing?
- Answer: Transformer-based models are the state-of-the-art architectures used in modern ASR systems. The transformer-based models produce the same or better results as other architectures but use significantly fewer parameters.
- Question: What is the best neural network algorithm for acoustic modelling?
- Answer: A combination of CNN architectures and transformer-based architectures produce the best results for ASR acoustic model characteristics.
- Question: Which audio feature extraction methods are the most efficient for natural language processing?
- Answer: Mel frequency cepstral coefficients and delta features were found to produce the best accuracy for ASR models.
- Question: How accurate does an automated speech recognition model have to be, to increase efficiency and reduce human input required?

- Answer: No ASR model will predict speech with a 0% WER. ASR models with WERs of less than 20% produce text that is 80% or more accurate. This would increase the efficiency as the human input required would be to correct the proportion of text that was predicted incorrectly.
- Question: Will a small transformer-based model produce a word error rate performance, that is comparable and similar to larger modern automated speech recognition systems?
- Answer: The small end-to-end transformer-based ASR model produced WER accuracies that were not as accurate as modern larger ASR models.
- Question: Is it possible to build an accurate ASR model with a limited amount of speech data?
- Answer: It is possible to build an ASR model with a limited amount of speech data, but if the data is too little, the model is restricted to the data rather than the model size and parameters and will therefore not be as accurate as modern ASR models.

6.2 ASR MODEL OUTPUT

The final end-to-end transformer-based ASR model designed in the research consisted of a CNN layer and a 2-headed transformer with three encoder layers and three decoder layers. The data used to train the model was the 2000 hours of Mozilla Common Voice 7.0 training data and the 1000 hours of LibriSpeech training data. The audio data was transformed to 32 MFCC and MFCC delta features as the input to the model. The output of the model was 30 character-based classes. The output of the model was processed by a small 4-gram language model to improve word understanding as it is a character-based ASR model.

The model produced a CER and WER of 27.89% and 54.5% respectively for the very noisy Common Voice testing dataset. The model also produced a CER and WER of 6.40% and 16.03% respectively for the LibriSpeech test clean dataset. A CER of 6.4% implies that 93.6% of the characters were predicted correctly for the test clean dataset and WER of 16.03% implies that 83.97% of the words were predicted correctly for the test clean dataset. This provides sufficient evidence that the small-scale ASR model is a viable speech recognition model.

6.3 SHORTCOMINGS

The final end-to-end transformer-based ASR model designed had a few limitations that caused the model not to be as accurate as modern ASR systems. The maximum available VRAM for the model was selected to be 12 GB as the goal of the model was to be small-scaled for use cases where industrial scale computer resources are not available. Another limitation was the amount of available data for training of the small-scaled ASR model. The available data consisted out of 3000 hours of data.

Modern ASR systems are all trained on over 40 000 hours of voice data and have computational sizes that require over 120 GB of VRAM for training on cluster type servers. More data would have allowed the research to determine how close the results of the small-scale ASR model would be to much larger modern ASR models, using the same data.

The n-gram language model used was a small statistical language model with a limited vocabulary. Larger, more modern language models can be added to improve the WER given the good CER achieved by the end-to-end ASR model.

6.4 FUTURE DEVELOPMENT

The research only looked at a limited number of combinations of CNN and transformer architectures. Future development would include experimenting on different neural network architectures and different layers within the transformer architecture. The transformer architecture can also be increased to have more heads and more encoder layers and decoder layers if more VRAM is available for training. Different modern language models can also be implemented along with the character-based ASR model depending on the scale of the final ASR system.

The main goal of this research was to develop a small-scale ASR model for use cases where practitioners do not have access to industrial scale datasets and computer resources. The ASR model is a proof of concept for under resourced languages that simply do not have large corpuses available. The first step in future development would be to test the small-scale ASR model architecture on different under resourced languages with limited datasets.

The model was also designed to be small enough to create a portable standalone ASR device, that can be used as a communication tool, where there isn't necessarily an internet connection for Google Translate and larger speech recognition tools. The next step to achieve this goal would be to create a portable device that the trained small-scale ASR model could be loaded onto for real life use.

CHAPTER 7 CONCLUSION

A character-based small-scale transformer architecture was used to create an end-to-end automated speech recognition model that produces comparable results in performance and word error rate (WER) to large scale character-based ASR models. The model did not produce WERs that were significantly close to the WERS of large modern character-based ASR models, but this is due to the model in this research having over 150 times fewer parameters and being trained with up to 53 000 hours less voice data than the larger modern ASR systems. The model produced, was trained on the Common Voice and LibriSpeech training datasets, and produced a CER and WER of 27.89% and 54.5% respectively on the Common Voice testing data as well as a CER and WER of 6.40% and 16.03% respectively on the LibriSpeech test-clean dataset. The best current large ASR model produced a WER of 1.8% on the LibriSpeech test-clean dataset, but with a 317 million parameter model and a transformer-based language model consisting of over 400 million parameters. The model in this paper has 2.214 million parameters and a small 4-gram language model. The model was comparable to an older GRU-based model that produced a WER of 11.9% and 31.1% respectively on the same LibriSpeech dataset, but at a significantly larger computational cost, as the architecture had 52.5 million parameters, which is a factor of 24 times larger than the architecture in this paper.

The small-scale model could be improved by adding more transformer layers, but this would increase the total parameters of the model and require more video RAM for training. The larger modern transformer ASR systems outperform the small-scale transformer ASR model, but adding more training data should increase the performance of the small-scale model significantly. All the character-based end-to-end ASR models still require a small language model to ensure that the predicted characters are transformed into actual words. A small 4-gram language model decreased the WER by over 20% in the Common Voice English 7.0 testing data. The main goal of this paper was to develop a small-scale ASR model for use cases where practitioners do not have access to industrial scale datasets and computer

resources. The ASR model is a proof of concept for under resourced languages that fall under the category of not having large corpuses available. The model was also designed to be small enough to create a portable standalone ASR device, that can be used as a communication tool, where there isn't necessarily an internet connection for Google Translate and larger speech recognition tools.

REFERENCES

[1] A. Maas, Z. Xie, D. Jurafsky, and A. Y. Ng, "Lexicon-free conversational speech recognition with neural networks," in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2015, pp. 345–354.

[2] N. E. Groce, "People with Disabilities," in *Social Injustice and Public Health*, 2009, pp. 258–261.

[3] J. Enrique Garcia, A. Ortega, E. Lleida, T. Lozano, E. Bernues, and D. Sanchez, "Audio and text synchronization for TV news subtitling based on Automatic Speech Recognition," in *2009 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting*, 2009, pp. 1–6.

[4] X. Chen, X. Liu, Y. Wang, M. J. F. Gales, and P. C. Woodland, "Efficient Training and Evaluation of Recurrent Neural Network Language Models for Automatic Speech Recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 11, pp. 2146–2157, 2016.

[5] H. Liao, E. McDermott, and A. Senior, "Large scale deep neural network acoustic modeling with semi-supervised training data for YouTube video transcription," in *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, 2013, pp. 368–373.

[6] S. Li, Y. Akita, and T. Kawahara, "Semi-Supervised Acoustic Model Training by Discriminative Data Selection From Multiple ASR Systems' Hypotheses," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 9, pp. 1524–1534, 2016.

[7] Y. Long, Y. Li, S. Wei, Q. Zhang, and C. Yang, "Large-Scale Semi-Supervised Training in Deep Learning Acoustic Model for ASR," *IEEE Access*, vol. 7, pp. 133 615–133 627, 2019.

[8] B. Wu, K. Li, F. Ge, Z. Huang, M. Yang, S. M. Siniscalchi, and C.-H. Lee, "An End-to-End Deep Learning Approach to Simultaneous Speech Dereverberation and Acoustic Modeling for Robust Speech Recognition," *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 8, pp. 1289–1300, 2017.

[9] B. Li, E. Zhou, B. Huang, J. Duan, Y. Wang, N. Xu, J. Zhang, and H. Yang, "Large scale recurrent neural network on GPU," in *2014 International Joint Conference on Neural Networks (IJCNN)*, 2014, pp. 4062–4069.

[10] W. Williams, N. Prasad, D. Mrva, T. Ash, and T. Robinson, "Scaling recurrent neural network language models," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, pp. 5391–5395.

[11] M. Lippi, M. A. Montemurro, M. Degli Esposti, and G. Cristadoro, "Natural language statistical features of LSTM-generated texts," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3326–3337, 2019.

[12] M. Sundermeyer, H. Ney, and R. Schlüter, "From feedforward to recurrent LSTM neural networks for language modeling," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 23, no. 3, pp. 517–529, 2015.

[13] T. Brants, A. C. Popat, P. Xu, F. J. Och, and J. Dean, "Large language models in machine translation," in *EMNLP-CoNLL 2007 - Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2007, pp. 858–867.

[14] 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*, vol. 30, pp. 5999–6009, Jun 2017.

[15] M. T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Conference Proceedings - EMNLP 2015: Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1412–1421.

[16] 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 - ICML '06*, vol. 148, 2006, pp. 369–376.

[17] R. Vergin, D. O'Shaughnessy, and A. Farhat, "Generalized mel frequency cepstral coefficients for large-vocabulary speaker-independent continuous-speech recognition," *IEEE Transactions on speech and audio processing*, vol. 7, no. 5, pp. 525–532, 1999.

[18] Mozilla, "Common Voice Corpus 7.0," 2021. [Online]. Available: https://commonvoice.mozilla.org/en/datasets (accessed: Aug 14, 2021)

[19] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an ASR corpus based on public domain audio books," in *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, 2015, pp. 5206–5210.

[20] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "Wav2Vec 2.0: A framework for self-supervised learning of speech representations," *Advances in neural information processing systems*, vol. 33, pp. 12 449–12 460, Jun 2020.

[21] R. Medar, V. S. Rajpurohit, and B. Rashmi, "Impact of training and testing data splits on accuracy of time series forecasting in machine learning," in *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)*, 2017, pp. 1–6.

[22] Q. Xiao, M. Qin, P. Guo, and Y. Zhao, "Multimodal fusion based on LSTM and a couple conditional hidden Markov model for Chinese sign language recognition," *IEEE Access*, vol. 7, pp. 112 258–112 268, 2019.

[23] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of NAACL-HLT*, 2019, pp. 4171–

4186.

[24] S. F. Chen, D. Beeferman, and R. Rosenfeld, "Evaluation metrics for language models," in *Proceedings of the DARPA Broadcast News Transcription and Understanding Workshop*, 1998, pp. 275– 280.

[25] J. Drexler and J. Glass, "Subword regularization and beam search decoding for end-to-end automatic speech recognition," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 6266–6270.

[26] Y. Zhang, R. Togneri, and M. Alder, "Phoneme-based vector quantization in a discrete HMM speech recognizer," *IEEE Transactions on Speech and Audio Processing*, vol. 5, no. 1, pp. 26–32, 1997.

[27] D. R. S. Caon, A. Amehraye, J. Razik, G. Chollet, R. V. Andreão, and C. Mokbel, "Experiments on acoustic model supervised adaptation and evaluation by K-Fold Cross Validation technique," in *2010 5th International Symposium On I/V Communications and Mobile Network*, 2010, pp. 1–4.

[28] D. Povey, D. Kanevsky, B. Kingsbury, B. Ramabhadran, G. Saon, and K. Visweswariah, "Boosted MMI for model and feature-space discriminative training," in *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2008, pp. 4057–4060.

[29] P. Polur and G. Miller, "Experiments with fast Fourier transform, linear predictive and cepstral coefficients in dysarthric speech recognition algorithms using hidden Markov model," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 13, no. 4, pp. 558–561, 2005.

[30] P. von Platen, C. Zhang, and P. Woodland, "Multi-span acoustic modelling using raw waveform signals," in *Interspeech 2019*, 2019, pp. 1393–1397.

[31] D. Deshwal, P. Sangwan, and D. Kumar, "Feature extraction methods in language identification: A survey," *Wireless Personal Communications*, vol. 107, no. 4, pp. 2071–2103, Aug 2019.

[32] A. Romero, "How Does Automated Closed Captioning Work?" May 2018. [Online]. Available: https://blog.video.ibm.com/ai-video-technology/how-does-automated-closed-captioning-work/ (accessed: Aug 7, 2020)

[33] C.-F. Juang, C.-T. Chiou, and H.-J. Huang, "Noisy speech recognition by hierarchical recurrent neural fuzzy networks," in *2005 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 5, 2005, pp. 5122–5125.

[34] K. Heafield, "KenLM: Faster and smaller language model queries," in *Proceedings of the sixth workshop on statistical machine translation*, 2011, pp. 187–197.

[35] P. Aleksic, M. Ghodsi, A. Michaely, C. Allauzen, K. Hall, B. Roark, D. Rybach, and P. Moreno, "Bringing contextual information to Google speech recognition," in *Interspeech 2015*, 2015, pp. 468–472.

[36] A. Baevski and M. Auli, "Adaptive input representations for neural language modeling," 2019, *arXiv:1809.10853*.

[37] T. Mikolov, S. Kombrink, L. Burget, J. Černockỳ, and S. Khudanpur, "Extensions of recurrent neural network language model," in *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, 2011, pp. 5528–5531.

[38] B. Roark, "A Survey of Discriminative Language Modeling Approaches for Large Vocabulary Continuous Speech Recognition," in *Automatic Speech and Speaker Recognition*. Chichester, UK: John Wiley & Sons, Ltd, 2009, pp. 115–137.

[39] K. Hwang and W. Sung, "Character-level incremental speech recognition with recurrent neural networks," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 2016-May, 2016, pp. 5335–5339.

[40] L. Dong, S. Xu, and B. Xu, "Speech-Transformer: A No-Recurrence Sequence-to-Sequence Model for Speech Recognition," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 5884–5888.

[41] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu *et al.*, "Conformer: Convolution-augmented transformer for speech recognition," in *INTERSPEECH, ISCA*, 2020, p. 5036–5040.

[42] G. I. Winata, S. Cahyawijaya, Z. Lin, Z. Liu, and P. Fung, "Lightweight and Efficient End-To-End Speech Recognition Using Low-Rank Transformer," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 6144–6148.

[43] N.-Q. Pham, T.-S. Nguyen, J. Niehues, M. Müller, S. Stüker, and A. Waibel, "Very Deep Self-Attention Networks for End-to-End Speech Recognition," 2019, *arXiv:1904.13377*.

[44] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," in *3rd International Conference on Learning Representations, ICLR San Diego, CA, USA*, 2015, pp. 1–16.

[45] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[46] K. Nassar, "Transformer-based language modeling and decoding for conversational speech recognition," 2020, *arXiv:2001.01140*.

[47] W.-N. Hsu, B. Bolte, Y.-H. H. Tsai, K. Lakhotia, R. Salakhutdinov, and A. Mohamed, "Hubert: Self-supervised speech representation learning by masked prediction of hidden units," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 29, pp. 3451–3460, 2021.

[48] Z. Min and J. Wang, "Exploring the integration of large language models into automatic speech recognition systems: An empirical study," *arXiv preprint arXiv:2307.06530*, 2023.

[49] Y. Kubo, S. Karita, and M. Bacchiani, "Knowledge transfer from large-scale pretrained language models to end-to-end speech recognizers," in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 8512–8516.

[50] K. Lakhotia, E. Kharitonov, W.-N. Hsu, Y. Adi, A. Polyak, B. Bolte, T.-A. Nguyen, J. Copet, A. Baevski, A. Mohamed *et al.*, "On generative spoken language modeling from raw audio," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 1336–1354, 2021.

[51] W.-C. Huang, C.-H. Wu, S.-B. Luo, K.-Y. Chen, H.-M. Wang, and T. Toda, "Speech recognition by simply fine-tuning BERT," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 7343–7347.

[52] C. Wang, Y. Tang, X. Ma, A. Wu, S. Popuri, D. Okhonko, and J. Pino, "Fairseq S2T: Fast Speech-to-Text Modeling with fairseq," 2020, *arXiv:2010.05171*.

[53] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli, "Fairseq: A Fast, Extensible Toolkit for Sequence Modeling," 2019, *arXiv:1904.01038*.

[54] G. Synnaeve, Q. Xu, J. Kahn, T. Likhomanenko, E. Grave, V. Pratap, A. Sriram, V. Liptchinsky, and R. Collobert, "End-to-end ASR: From supervised to semi-supervised learning with modern architectures," 2019, *arXiv:1911.08460*.

[55] A. Baevski, W.-N. Hsu, A. Conneau, and M. Auli, "Unsupervised speech recognition," *Advances in Neural Information Processing Systems*, vol. 34, pp. 27 826–27 839, May 2021.

[56] S. Schneider, A. Baevski, R. Collobert, and M. Auli, "Wav2Vec: Unsupervised Pre-Training for Speech Recognition," in *Interspeech 2019*, 2019, pp. 3465–3469.

[57] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *33rd International conference on machine learning, ICML 2016*, 2016, pp. 173–182.

[58] A. Conneau, A. Baevski, R. Collobert, A. Mohamed, and M. Auli, "Unsupervised cross-lingual representation learning for speech recognition," in *Interspeech 2021*, 2021, pp. 2426–2430.

[59] J. Grosman, "XLSR Wav2Vec2 English by Jonatas Grosman," 2021. [Online]. Available: https://huggingface.co/jonatasgrosman/wav2vec2-large-xlsr-53-english (accessed: Feb 11, 2022)

[60] A. Babu, C. Wang, A. Tjandra, K. Lakhotia, Q. Xu, N. Goyal, K. Singh, P. von Platen, Y. Saraf, J. Pino *et al.*, "XLS-R: Self-supervised cross-lingual speech representation learning at scale," in *Interspeech 2022*, 2022, pp. 2278–2282.

[61] H. Scheidl, S. Fiel, and R. Sablatnig, "Word Beam Search: A Connectionist Temporal Classification Decoding Algorithm," in *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, 2018, pp. 253–258.

[62] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2017, pp. 1–11.

ADDENDUM A : ACKNOWLEDGEMENTS

We thank the MultiChoice Chair of Machine Learning for sponsoring the funding for the research of this paper. The University of Pretoria EECE department provided the facilities and resources to complete the paper. We would also like to thank Mozilla for the publicly available Common Voice dataset and to OpenSLR and Vassil Panayotov for the publicly available LibriSpeech dataset.