



Automatic Speech Recognition for Afrikaans and isiXhosa

by
Lucas Meyer

Research assignment presented in partial fulfilment of the requirements for the degree of Master of Machine Learning and Artificial Intelligence in the Faculty of Applied Mathematics at Stellenbosch University.

Supervisor: Dr H. Kamper

November 2023

Acknowledgements

I would like to thank my supervisor, **Herman Kamper**, for allowing me to do this project with him. He has provided me with great advice and many helpful resources. He is also the creator of this report template.

Plagiaatverklaring / *Plagiarism Declaration*

1. Plagiaat is die oorneem en gebruik van die idees, materiaal en ander intellektuele eiendom van ander persone asof dit jou eie werk is.

Plagiarism is the use of ideas, material and other intellectual property of another's work and to present it as my own.

2. Ek erken dat die pleeg van plagiaat 'n strafbare oortreding is aangesien dit 'n vorm van diefstal is.

I agree that plagiarism is a punishable offence because it constitutes theft.

3. Ek verstaan ook dat direkte vertalings plagiaat is.


I also understand that direct translations are plagiarism.

4. Dienooreenkomstig is alle aanhalings en bydraes vanuit enige bron (ingesluit die internet) volledig verwys (erken). Ek erken dat die woordelike aanhaal van teks sonder aanhalingstekens (selfs al word die bron volledig erken) plagiaat is.

Accordingly all quotations and contributions from any source whatsoever (including the internet) have been cited fully. I understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism

5. Ek verklaar dat die werk in hierdie skryfstuk vervat, behalwe waar anders aangedui, my eie oorspronklike werk is en dat ek dit nie vantevore in die geheel of gedeeltelik ingehandig het vir bepunting in hierdie module/werkstuk of 'n ander module/werkstuk nie.

I declare that the work contained in this assignment, except where otherwise stated, is my original work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment.

| | |
|---|---|
| 22614524 Studentenommer / <i>Student number</i> |  Handtekening / <i>Signature</i> |
| L. Meyer Voorletters en van / <i>Initials and surname</i> | 6 November 2023 Datum / <i>Date</i> |

Abstract

Developing high-quality automatic speech recognition (ASR) systems for languages such as Afrikaans and isiXhosa remains a challenging task, mainly due to a limited number of available speech resources. In this study, we investigate the use of additional data from related languages in the development of ASR systems for Afrikaans and isiXhosa. Specifically, we determine whether the use of Dutch data improves the performance of our Afrikaans systems and whether the use of isiZulu data improves the performance of our isiXhosa systems. We believe that while the use of Dutch and isiZulu data may improve performance, it also requires more computational resources to train our systems.

We compare two training strategies. The first strategy involves training on only the target language, and the second strategy involves training on the target language and a related language. We train several Afrikaans and isiXhosa models using both strategies and evaluate the performance of each model on our own validation and test data. Additionally, we use separately trained language models to improve the performance of our best models. We find that using additional data from a related language can slightly improve performance. By only training on the target language, our best Afrikaans model achieves a 28.30% word error rate (WER) on our Afrikaans test data and our best isiXhosa model achieves a 41.47% WER on our isiXhosa test data. By using additional Dutch and isiZulu data, our best Afrikaans model achieves a 27.50% WER on our Afrikaans test data and our best isiXhosa model achieves a 40.51% WER on our isiXhosa test data. We believe that using additional data from related languages could potentially provide a strategy for developing ASR systems in other under-resourced settings. Code, examples, and models: <https://github.com/lucas-meyer-7/Speech-Recognition-Afrikaans-isiXhosa>.

Chapter 1

Introduction

Automatic speech recognition (ASR) is the task of identifying the spoken words for a given speech recording and returning the text of the spoken words. For example, given a recording of a speaker saying the sentence “The quick brown fox jumps over the lazy dog”, the goal of ASR is to predict each character in the sentence and to make as few mistakes as possible. ASR systems are used across many domains such as education [1], helping people with disabilities [2], and automatic captioning [3]. These systems are typically developed for well-resourced¹ languages such as English, Spanish, and Mandarin. However, developing ASR systems for under-resourced languages is a more challenging task. The size and speaker diversity of the data used to train an ASR system has a significant effect on the accuracy and generalization ability of the system. Therefore, more data and a greater variety of speakers typically improves the performance of a particular system. The focus of our study is performing ASR for two under-resourced languages: Afrikaans and isiXhosa. Furthermore, we investigate whether the use of additional speech data from a related language can improve the performance of our Afrikaans and isiXhosa ASR systems. We believe that the use of additional data could be a potential strategy to improve the generalization ability of our systems. However, a disadvantage of this strategy is that our system may confuse words from the target language with words from the related language. The other disadvantage is that this strategy requires more computational resources since the strategy involves training a model twice. We use additional Dutch data because of its relation with Afrikaans [4], and we use additional isiZulu data because of its relation with isiXhosa [5]. Our work is related to a study by Kamper et al. [6] which involves the use of North American speech resources for the development of South African English ASR systems.

The general approach for performing ASR has remained the same for several years: transform the speech data into a feature representation, then map the features to a sequence of characters which are merged to form the predicted text. Traditionally, hidden Markov models [7] and neural networks such as Listen, Attend and Spell (LAS) [8] have been used to perform ASR. In recent years, self-supervised learning techniques have become popular within the field of ASR and other speech-related tasks. Self-supervised learning techniques

¹A well-resourced language is a language for which there are extensive speech resources available.

allow for learning feature representations from unlabeled data, which is particularly useful in under-resourced settings. One of these techniques is wav2vec 2.0 [9], which is a neural network used to learn feature representations for speech using unlabeled speech data. A pre-trained wav2vec 2.0 model can be fine-tuned to a variety of speech-related tasks such as ASR, automatic speech translation, and speech classification. The issue is that pre-training wav2vec 2.0 with unlabeled data is too computationally expensive for the computational resources available to our study (2.4). Therefore, our approach for developing ASR systems for Afrikaans and isiXhosa involves fine-tuning pre-trained wav2vec 2.0 models for ASR using the connectionist temporal classification (CTC) algorithm. We use the XLS-R [10] model as our pre-trained model, which is a large-scale wav2vec 2.0 model trained on 128 different languages.

For our experimental setup, we consider two training strategies: *basic fine-tuning* and *sequential fine-tuning*. Basic fine-tuning involves fine-tuning on one language, and sequential fine-tuning involves fine-tuning on a related language and then on the target language. We create our own Afrikaans and isiXhosa speech datasets by combining recordings from three different datasets. We also use Dutch and isiZulu speech data from one of the three datasets. We train several Afrikaans and isiXhosa ASR models using both training strategies, and we test different hyperparameter values during the training of each model. We evaluate the performance of each model by computing the word error rate (WER) on our validation and test data. We select the best Afrikaans and isiXhosa model for each strategy and add a 5-gram language model (LM) with Kneser-Ney smoothing. Our LMs are separately trained on scraped Afrikaans and isiXhosa Wikipedia data. Using an LM for ASR typically improves performance by mitigating simple spelling mistakes that occur when decoding with CTC.

In our experimental results, we find that sequential fine-tuning results in slightly better performance for both our Afrikaans and isiXhosa models. By using sequential fine-tuning, our best Afrikaans model achieves a 27.50% WER on our Afrikaans test data and our best isiXhosa model achieves a 40.51% WER on our isiXhosa test data.

Our main contribution is a comparison of training strategies for the development of ASR systems in under-resourced settings, specifically for Afrikaans and isiXhosa. Furthermore, since both strategies only require a single GPU for fine-tuning XLS-R on our datasets, anyone with access to Google Colab can utilize our code to train their own Afrikaans or isiXhosa ASR model. However, the generalization ability of our best-performing models is still limited. We believe that in future work, pre-training on a large dataset of unlabeled speech recordings may improve the generalization ability of our models.

The report is organized as follows. We first present the background of ASR and our explored tasks in Chapter 2. We discuss our datasets and provide an outline of our experimental setup in Chapter 3. Our experimental results are summarized and discussed in Chapter 4. Finally, we conclude our study in Chapter 5.

Chapter 2

Background

This chapter provides an overview of automatic speech recognition (ASR), the general approach of ASR, and the common challenges of ASR. We provide a detailed explanation of wav2vec 2.0, a framework for learning speech feature representations. We discuss how the connectionist temporal classification (CTC) algorithm is used to fine-tune speech feature representations for ASR. The chapter concludes with an explanation of why pre-training is infeasible for our study, and a brief discussion of a pre-trained wav2vec 2.0 model called XLS-R.

2.1. The automatic speech recognition task

Automatic speech recognition (ASR), also known as speech recognition, is the task of identifying the spoken words for a given speech recording and returning the text, or *transcription*, of the spoken words. For example, given a recording of a speaker saying the sentence “The quick brown fox jumps over the lazy dog”, the goal of ASR is to predict each character in the sentence and to make as few mistakes as possible.

The general approach for ASR involves two steps. In the first step, we transform the speech data into a feature representation which we refer to as *speech features* for the remainder of the report. In the second step, we use supervised learning techniques to map speech features to a sequence of characters that are merged into a sentence.

We compute speech features because speech data is difficult to interpret. Speech data, which is a form of audio, consists of a sequence of floating point numbers called *samples*. The sequence of samples represents amplitude measurements of the recorded sound wave. Note that sound is a continuous wave function, but a recording is a discretized version of the original sound (see Figure 2.1). The issue is that mapping a sequence of samples to a sequence of characters is difficult. Therefore, it is common in ASR and other speech-related tasks to transform speech data into speech features. The traditional approach is to compute speech features by transforming the speech data from the amplitude-time domain to the frequency-time domain using the fast Fourier transform (FFT) algorithm [11], [12].

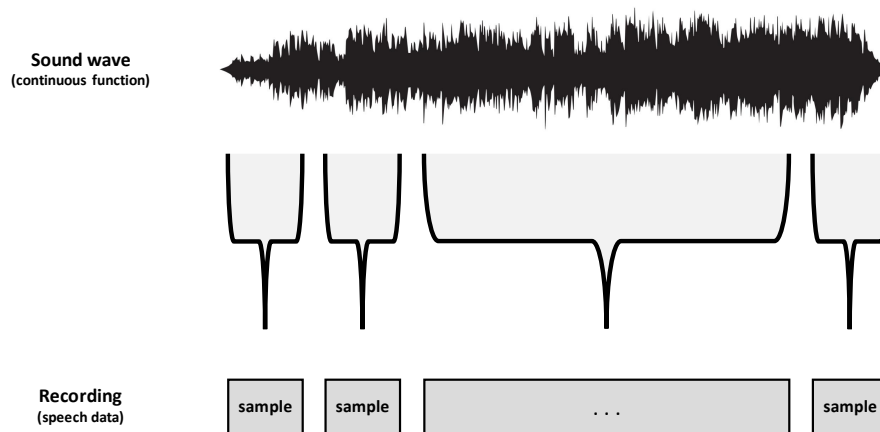


Figure 2.1: A diagram that explains what speech data (audio data) represents. Note that this is an oversimplified explanation. The intent of the diagram is to explain why it is difficult to extract meaning directly from speech data.

In this study, we discuss an approach to obtain speech features using self-supervised learning. Self-supervised learning is a machine learning paradigm in which unlabeled data is used for learning, similar to unsupervised learning. Self-supervised learning differs from unsupervised learning in that it leverages the unlabeled data to create artificial labels, often through tasks designed to predict part of the input from other parts. For instance, in the context of speech features, a task of the model may be to predict the next sample of an audio sequence, using the previous samples as context. This approach transforms the problem into a supervised problem without the need for explicit labels. Before explaining this approach, we discuss the importance of constructing good ASR datasets.

2.1.1. Automatic speech recognition data

The first step of creating an ASR model is to prepare speech datasets for training, evaluating, and testing the model. A single dataset entry consists of a speech recording (of a spoken sentence) and the corresponding text of the spoken sentence. The following paragraphs demonstrate why preparing each dataset has a significant effect on the accuracy and generalization performance of ASR models.

Exclusivity around partitioning data. The validation and test data should not contain recordings of voices that also appear in the training data. This is to ensure that the model is not overfitting to the voices in the training data.

The amount of training data. Using more training data often leads to better accuracy and generalization performance of ASR models. A small dataset with few unique speakers may lead to overfitting of the speakers (voices) in the dataset.

Voice diversity. The training data should contain a wide variety of speakers. The accent of a speaker, which depends on the gender, age, and ethnicity of the speaker, may influence the generalization ability of ASR models. Generally, male voices have a lower pitch than female voices, and adults have a lower pitch than children.

The difference between read speech and conversational speech. The training data should contain both read speech and conversational speech. Speakers tend to speak more clearly when reading text from a transcript, compared to speakers talking to each other in conversation. According to Jurafsky et al. [13], recent ASR models obtain low error rates for recordings of read speech. However, performing ASR for recordings of conversational speech remains a challenge.

In the following section, we discuss wav2vec 2.0, which is the technique used in this study to extract speech features.

2.2. wav2vec 2.0

wav2vec 2.0 [9] is a machine learning model used for learning speech features using unlabeled¹ speech data. wav2vec 2.0 can be applied to a variety of speech-related tasks such as ASR, automatic speech translation, and speech classification. It proves to be particularly useful in cases where a lot of unlabeled data is available, but not a lot of labeled data. For ASR, the authors show that using just ten minutes of labeled data and pre-training on 53k hours of unlabeled data achieves 4.8/8.2 WER (3.3) on the clean/other test data of the Librispeech [14] dataset.

The general two-step approach for using wav2vec 2.0 for any speech-related task is the following. In the first step, the wav2vec 2.0 model is trained (or “pre-trained”) using unlabeled data, which gives you a model that transforms speech data into speech features. In the second step, the model is fine-tuned on a downstream task using labeled data. Fine-tuning wav2vec 2.0 (2.2.5) for ASR involves replacing the head of the pre-trained model with an appropriate loss function such as CTC (2.3).

The wav2vec 2.0 architecture is illustrated by the diagram in Figure 2.2. There are three important components of the architecture: the feature encoder, the quantization module, and the context network. The objective of wav2vec 2.0 becomes clear only after understanding each of the three components. Therefore, we explain how to pre-train wav2vec 2.0 after explaining the three components in detail.

¹Unlabeled speech data simply refers to recordings without transcription text (labels).

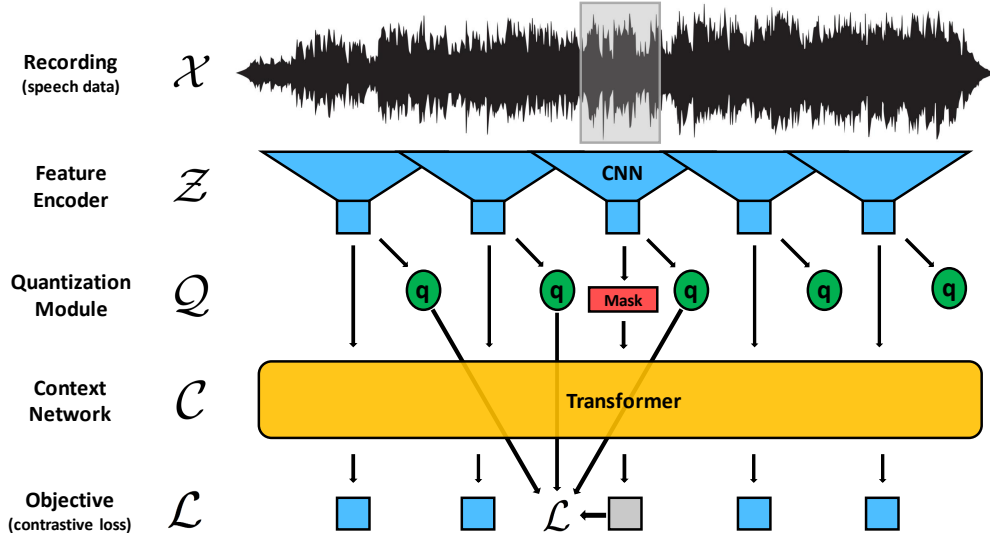


Figure 2.2: A diagram of the network architecture of wav2vec 2.0. This diagram is based on Figure 1 of the wav2vec 2.0 paper [9].

2.2.1. Feature encoder

The feature encoder maps speech data to latent representations: $f : \mathcal{X} \rightarrow \mathcal{Z}$. In other words, the feature encoder f maps a sequence of audio samples $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}$ into a sequence of latent feature vectors $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(t)}$.

The speech data is scaled to have zero mean and unit variance before being fed into the feature encoder. The feature encoder consists of seven convolutional blocks, where each convolutional block contains a temporal convolutional layer, layer normalization [15], and the GELU [16] activation function.

Each temporal convolutional layer contains 512 channels, the strides of the seven temporal convolutional layers are $(5, 2, 2, 2, 2, 2, 2)$, and the kernel widths are $(10, 3, 3, 3, 3, 2, 2)$. This results in each $\mathbf{z}^{(t)}$ representing 25 ms of audio (or 400 input samples), strided by about 20 ms. Layer normalization scales the logits after each convolutional layer to have zero mean and unit variance, which has been shown to increase the chances of earlier convergence. GELU is an activation function recently used for several NLP-related tasks.

2.2.2. Quantization module

The quantization module maps the latent feature vectors into discrete speech units: $h : \mathcal{Z} \rightarrow \mathcal{Q}$. Unlike written language, which can be discretized into tokens such as characters or sub-words, speech does not have natural sub-units [17]. This is because speech is a sound wave, which is a continuous function of time. The quantization module is a method in which discrete speech units are automatically learned using product quantization.

To perform product quantization, the quantization module uses G *codebooks*, where each codebook contains V *codebook entries* $\mathbf{e}_1, \dots, \mathbf{e}_V$. The following steps describe the process of automatically assigning a discrete speech unit to each latent feature vector $\mathbf{z}^{(t)}$:

1. Transform $\mathbf{z}^{(t)}$ into $\mathbf{l}^{(t)} \in \mathbb{R}^{G \times V}$ using a linear transformation (feed-forward neural network).
2. Choose one codebook entry \mathbf{e}_g for each codebook $g = 1, \dots, G$, based on the values of $\mathbf{l}^{(t)}$.
3. Concatenate the codebook entries $\mathbf{e}_1, \dots, \mathbf{e}_G$.
4. Transform the resulting vector into $\mathbf{q}^{(t)} \in \mathbb{R}^f$ using another linear transformation (feed-forward neural network).

The two linear transformations are feed-forward neural networks $\text{FF}_1 : \mathbb{R}^f \rightarrow \mathbb{R}^{G \times V}$ and $\text{FF}_2 : \mathbb{R}^d \rightarrow \mathbb{R}^f$. In the second step above, the codebook entry \mathbf{e}_g is chosen as the one with the arg max of the logits \mathbf{l} . Choosing the codebook entries in this way is non-differentiable. Fortunately, we can use the Gumbel softmax to choose codebook entries in a fully differentiable way. \mathbf{e}_g is chosen as the entry that maximizes

$$p_{g,v} = \frac{\exp(\mathbf{l}_{g,v}^{(t)} + n_v)/\tau}{\sum_{k=1}^V \exp(\mathbf{l}_{g,k}^{(t)} + n_k)/\tau}, \quad (2.1)$$

where τ is a non-negative temperature, $n = -\log(-\log(u))$, and u are uniform samples from $\mathcal{U}(0, 1)$. During the forward pass, the codeword i is chosen by $i = \text{argmax}_j p_{g,j}$, and during the backward pass, the true gradient of the Gumbel softmax outputs is used.

2.2.3. Context network

The context network maps the latent feature vectors into contextualized representations: $g : \mathcal{Z} \rightarrow \mathcal{C}$. The main component of the context network is a Transformer encoder [18]. Due to the popularity of Transformers, we omit the details of the Transformer architecture. However, we recommend illustrated guides such as [19] for more information about the Transformer architecture.

The following steps describe how the latent feature vectors are processed before being fed into the Transformer encoder.

1. The latent feature vectors are fed into a *feature projection layer* to match the model dimension of the context network.
2. Positional embedding vectors are added to the inputs using *relative positional encoding* [20] instead of absolute positional encoding. The relative positional encoding is implemented using grouped convolution [21].

3. Inputs are fed into the GELU activation function, followed by layer normalization.

The output after processing the latent feature vectors is fed into the Transformer encoder, which results in a sequence of contextualized representations $\mathbf{c}^{(1)}, \dots, \mathbf{c}^{(T)}$. The architectural details of the Transformer encoder for the LARGE version of wav2vec 2.0 are:

- Number of Transformer blocks: $B = 24$.
- Model dimension: $H_m = 1024$.
- Inner dimension: $H_{ff} = 4096$.
- Number of attention heads: $A = 16$.

2.2.4. Pre-training with wav2vec 2.0

Here we discuss how the three components are used to perform pre-training with wav2vec 2.0. The main objective of pre-training is a contrastive task, which involves masking a proportion of the output of the feature encoder $(\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(t)})$.

A proportion of the latent feature vectors are masked and replaced by a shared feature vector \mathbf{z}_M , before being fed into the context network. Note, that the inputs to the quantization module are not masked. A proportion p of starting indices in $\{1, \dots, t\}$ are randomly sampled. Then, for each starting index i , consecutive M time steps are masked (where spans may overlap).

The dimensionality of the output of the context network $(\mathbf{c}^{(1)}, \dots, \mathbf{c}^{(T)})$ matches the dimensionality of the output of the quantization module $(\mathbf{q}^{(1)}, \dots, \mathbf{q}^{(T)})$. The contrastive task involves predicting the corresponding quantized target $\mathbf{q}^{(t)}$ for a context vector $\mathbf{c}^{(t)}$. Additionally, 100 negative distractors are uniformly sampled for each masked position. The model is encouraged to minimize the distance (cosine similarity) between $\mathbf{q}^{(t)}$ and $\mathbf{c}^{(t)}$, and to maximize the distance between $\mathbf{q}^{(t)}$ and the negative distractors. We provide an explanation of the training objective below.

Training objective. There are two objectives (loss functions) that wav2vec 2.0 optimizes simultaneously. The first loss function is the contrastive loss \mathcal{L}_m which encourages the model to identify the true quantized representation for a masked time step within a set of distractors. The second loss function is the diversity loss \mathcal{L}_d which encourages the model to equally use the codebook entries from the quantization module. The full training objective is given by $\mathcal{L} = \mathcal{L}_m + \alpha \mathcal{L}_d$, where α is a tuned hyperparameter.

Contrastive loss. The contrastive loss is responsible for training the model to predict the correct quantized representation \mathbf{q}_t from a set of candidate representations $\tilde{\mathbf{q}} \in \mathcal{Q}_t$. The set \mathcal{Q}_t includes the target \mathbf{q}_t and K distractors sampled uniformly from other masked time steps. The contrastive loss is given by

$$\mathcal{L}_m = -\log \frac{\exp(\text{sim}(\mathbf{c}_t, \mathbf{q}_t)\kappa)}{\sum_{\tilde{\mathbf{q}} \in \mathcal{Q}_t} \exp(\text{sim}(\mathbf{c}_t, \tilde{\mathbf{q}})\kappa)}, \quad (2.2)$$

where κ represents a constant temperature, and $\text{sim}(a, b)$ denotes the cosine similarity between context representation c_t and quantized representations q . This loss encourages the model to assign high similarity to the true positive target and penalize high similarity with negative distractors.

Diversity loss. The diversity loss is a regularization technique aimed at promoting the equal use of codebook entries. It is based on entropy and is calculated as:

$$\mathcal{L}_d = \frac{1}{GV} \sum_{g=1}^G -H(\bar{p}_g) = -\frac{1}{GV} \sum_{g=1}^G \sum_{v=1}^V \bar{p}_{g,v} \log \bar{p}_{g,v}. \quad (2.3)$$

This loss maximizes the entropy of the softmax distribution $\bar{p}_{g,v}$ over codebook entries, encouraging the model to utilize all code words equally.

2.2.5. Fine-tuning wav2vec 2.0 for automatic speech recognition

To fine-tune a pre-trained wav2vec 2.0 model for ASR, it is required to prepare a labeled dataset. Typically, the head of the model is replaced with a linear layer that has an equal number of output neurons as the size of the vocabulary² of the labeled dataset. Finally, the model is optimized using a supervised learning algorithm called connectionist temporal classification.

2.3. Connectionist temporal classification

Connectionist temporal classification (CTC) [22] is an algorithm developed to map a sequence of speech features to a sequence of characters. Our explanation of CTC is heavily based on the speech recognition and text-to-speech chapter in [13]. We recommend readers to use Figure 2.3 as a visual aid for our explanation.

Given a sequence of speech features, CTC maps each speech feature vector to a single character, which results in a sequence of characters known as an *alignment*. Then, a *collapsing function* is used to merge consecutive duplicate characters in the alignment. The

²The vocabulary refers to the set of unique characters contained in the transcriptions (labels) of the dataset.

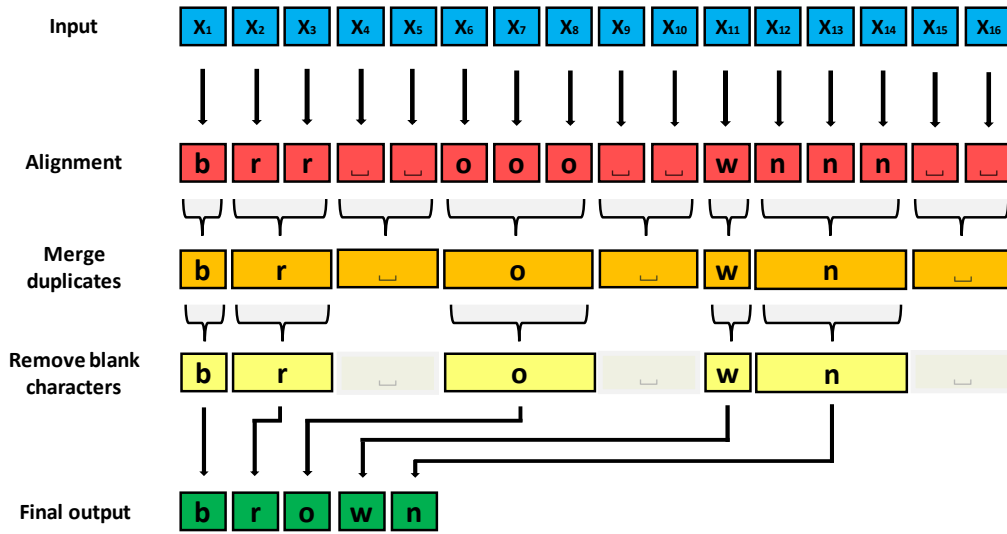


Figure 2.3: A diagram that describes the alignment procedure of CTC. This diagram is based on Figure 16.8 of [13].

authors of CTC propose the use of a special character called a *blank*, which is represented by “_”. The blank character accounts for words that contain consecutive duplicate characters (such as the word “dinner”, which contains two consecutive “n” characters). With the addition of the blank character, the collapsing function is responsible for merging consecutive duplicate characters and removing all instances of blank characters. Following the notation of [13], we define the collapsing function as a mapping $B : A \rightarrow Y$ for an alignment A and output text Y . Notice that B is many-to-one since many different alignments can map to the same output text (see Figure 2.4).

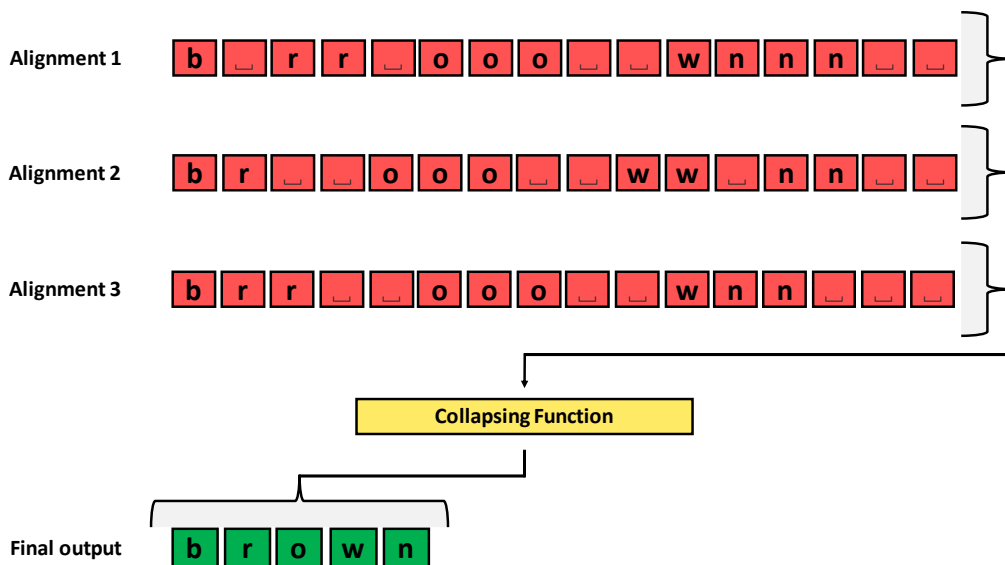


Figure 2.4: Three different alignments that produce the same output text when using the CTC collapsing function. This diagram is based on Figure 16.9 of [13].

In [13], the set of all alignments that map to the same output text Y is denoted as $B^{-1}(Y)$, where B^{-1} is the inverse of the collapsing function. This notation is useful for our explanation of CTC inference and CTC training.

2.3.1. Connectionist temporal classification inference

Here we discuss how CTC models the probability of the output text Y for a sequence of speech features $X = \{x^{(1)}, \dots, x^{(T)}\}$, denoted by $P_{\text{CTC}}(Y|X)$. The conditional probability above can be expressed as the summation over the probabilities of all possible alignments that produce the output text Y . Hence, we obtain the expression:

$$P_{\text{CTC}}(Y|X) = \sum_{A \in B^{-1}(Y)} P(A|X). \quad (2.4)$$

We still need an expression for $P(A|X)$. To compute $P(A|X)$, CTC makes a strong conditional independence assumption. It assumes that each alignment character $a^{(t)} \in \{a^{(1)}, \dots, a^{(T)}\}$ is computed independently of all the other alignment characters. Hence, we obtain the expression:

$$P(A|X) = \prod_{t=1}^T p(a^{(t)}|x^{(t)}). \quad (2.5)$$

Now we can find the best possible alignment for a given X , by simply choosing the most probable character at each time step. We compute each alignment character $\{a^{(1)}, \dots, a^{(T)}\}$, apply the collapsing function, and obtain the output text Y . However, there is an issue with the greedy approach described above. The issue is that the most probable output text \hat{Y} may not correspond with the most probable alignment sequence $\{\hat{a}^{(1)}, \dots, \hat{a}^{(T)}\}$. The reason for this is that many possible alignments lead to the same output text. Therefore, the most probable output text \hat{Y} , for a given X , corresponds to the maximum summation of the probabilities of all its possible alignments:

$$\hat{Y} = \operatorname{argmax}_Y P_{\text{CTC}}(Y|X) = \operatorname{argmax}_Y \left(\sum_{A \in B^{-1}(Y)} P(A|X) \right) \quad (2.6)$$

$$= \operatorname{argmax}_Y \left(\sum_{A \in B^{-1}(Y)} \prod_{t=1}^T p(a^{(t)}|x^{(t)}) \right). \quad (2.7)$$

There are many possible alignments, and summing over all the possible alignments is expensive and infeasible. We can use dynamic programming to approximate this sum, by using a modified version of Viterbi beam search [23]. The beam search returns a user-specified number of potential output texts, and a loss function is used to score each output text. The text with the best score is chosen as the final prediction.

2.3.2. Connectionist temporal classification training

The CTC loss function for a dataset D is equal to the summation of the negative log-likelihoods of the correct output Y for each corresponding input X :

$$L_{\text{CTC}} = \sum_{(X,Y) \in D} -\log P_{\text{CTC}}(Y|X) \quad (2.8)$$

$$= \sum_{(X,Y) \in D} -\log \left(\sum_{A \in B^{-1}(Y)} \prod_{t=1}^T p(a^{(t)}|x^{(t)}) \right). \quad (2.9)$$

Once again, we cannot compute a summation over all the possible alignments. The summation is efficiently computed using a variant of the *forward-backward algorithm*. An explanation of the forward-backward algorithm used for training CTC is given in [23].

2.3.3. Improving connectionist temporal classification with a language model.

Because of the strong conditional independence assumption mentioned earlier, CTC does not implicitly learn a language model (LM) over the data. Therefore, the CTC algorithm commonly predicts sentences that contain obvious spelling mistakes. We can mitigate this issue by using a separately trained LM. This is done by adding additional terms to the CTC loss function with interpolation weights:

$$\text{score}_{\text{CTC}}(Y|X) = \log(P_{\text{CTC}}(Y|X)) + \lambda_1 \log(P_{\text{LM}}(Y)) + \lambda_2 L(Y), \quad (2.10)$$

where λ_1 and λ_2 are the interpolation weights. $P_{\text{LM}}(Y)$ is the probability of the LM and $L(Y)$ is a length factor. In this study, we use n -gram LMs with Kneser-Ney smoothing [24], [25]. We use 5-gram LMs, due to their popularity in recent ASR literature. We omit the details of n -grams and Kneser-Ney smoothing.

We have now discussed how to create an ASR model that is pre-trained using wav2vec 2.0 and fine-tuned using CTC. However, in the following section, we discuss an issue with this approach for our particular study.

2.4. The difficulty of pre-training using wav2vec 2.0

Pre-training with wav2vec 2.0 using unlabeled data is infeasible for our study because:

1. Pre-training with wav2vec 2.0 is known to be quite unstable³.
2. It requires 8 GPU V100s with 16 GB RAM each, and approximately 7 days, to finish pre-training a large-scale wav2vec 2.0 model on a standard-size dataset [26].

³For reference, see Note 1 of [26].

At the time of conducting this study, we do not have access to the computational resources required to perform pre-training experiments. Therefore, our study focuses on fine-tuning already pre-trained wav2vec 2.0 models, rather than performing pre-training and then fine-tuning. In this section, we discuss XLS-R, which is the pre-trained wav2vec 2.0 model used in this study.

2.4.1. XLS-R

XLS-R [10] is a large-scale wav2vec 2.0 model trained on 128 different languages to learn *cross-lingual* speech features. Since XLS-R is a pre-trained wav2vec 2.0 model, it can be *fine-tuned* (2.2.5) on a wide range of downstream tasks such as ASR, automatic speech translation, and speech classification. XLS-R achieves better performance than the previous state-of-the-art in several benchmark tests for all three downstream tasks.

The XLS-R model attempts to build better speech features for low-resource languages by leveraging cross-lingual transfer from high-resource languages such as English. The model is trained using batches that contain samples from multiple languages L . Batches are sampled from a probability distribution $p_l \sim \left(\frac{n_l}{N}\right)^\alpha$ where:

- $l \in \{1, \dots, L\}$ represents each language.
- N is the number of hours of all the unlabeled data and n_l is the number of hours of unlabeled data for the language l .
- α controls how much data is chosen from the high-resource and low-resource languages. If $\alpha = 0$, then there is an equal probability of selecting from each of the 128 languages.

The batch selection strategy results in diverse batches that contain recordings from different languages, which encourages the model to learn cross-lingual speech features. This strategy mitigates the case where a batch contains only recordings from one language (which encourages the model to learn mono-lingual speech features and slows down training).

That concludes the background chapter of this report. Firstly, we discussed the general approach of ASR and the common challenges of developing ASR systems. Secondly, we explained wav2vec 2.0 which is a neural network that learns (using unlabeled data) how to transform speech data into speech features. Thirdly, we explained how the CTC algorithm is used to fine-tune wav2vec 2.0 speech features for ASR. Finally, we discussed XLS-R which is a large-scale pre-trained wav2vec 2.0 model used in this study.

Chapter 3

Experimental Setup

This chapter provides our research question, and how we attempt to answer this question using the results of different experiments. We discuss the speech data used for our ASR models, as well as the text data used for our language model (LM). The chapter ends with a discussion of the metric used to evaluate our ASR models.

3.1. Research question and experiments

The focus of our research is performing ASR for Afrikaans and isiXhosa with limited speech data and computational resources. Our main research question is determining whether the use of additional Dutch and isiZulu speech data can help develop ASR systems for Afrikaans and isiXhosa.

Our approach involves fine-tuning (2.2.5) the **XLS-R (300M)** model using different training strategies. Two training strategies are considered in this study. The first strategy is called *basic fine-tuning* and it involves fine-tuning XLS-R (300M) on one dataset (one language). The second strategy is called *sequential fine-tuning* and it involves fine-tuning XLS-R (300M) on two datasets (two languages) separately. The intent of sequential fine-tuning is to first fine-tune on the related language (e.g. Dutch) and then fine-tune on the target language (e.g. Afrikaans). We create several models using both training strategies, and each model is evaluated using the word error rate (WER) (3.3). We choose the best Afrikaans and isiXhosa model for both strategies based on which model has the lowest WER on our validation data. Finally, we add a 5-gram LM (2.3.3) to our best-performing models (to improve performance) and evaluate the models on our test data.

The following hyperparameters are tuned during the training of each model: **batch size**, **gradient accumulation steps**, **evaluation steps**, and **patience**. To clarify, **evaluation steps** refers to the number of steps between each evaluation on the validation data, and **patience** refers to the hyperparameter used for early stopping [27]. We use early stopping on the validation data to prevent our models from overfitting on the training data.

3.2. Data

We use three different speech datasets to create an **Afrikaans dataset** and an **isiXhosa dataset**, which contains transcribed speech data. We also use one of the datasets to create Dutch and isiZulu datasets. We discuss the three datasets in the paragraphs below.

NCHLT. The NCHLT [28] dataset contains transcribed speech data for the eleven official South African languages. The dataset contains approximately 200 speakers per language. The NCHLT recordings (on average) are the shortest of the three datasets, with most recordings being between 2 and 6 seconds. Based on a brief inspection, the transcription texts of this dataset contain very few words and rarely contain full sentences.

FLEURS. The FLEURS [29] dataset contains transcribed speech data for 102 different languages, including Afrikaans, Dutch, isiXhosa, and isiZulu. Each language has its own training, validation, and test data split. No information is given about the number of speakers for each language. The FLEURS recordings (on average) are the longest of the three datasets, with most recordings being between 7 and 20 seconds. Based on a brief inspection, the transcription texts of this dataset contain full sentences.

High-quality TTS. The High-quality text-to-speech (TTS) [30] dataset contains high-quality transcribed speech data for four South African languages: Afrikaans, Sesotho, Setswana and isiXhosa. There are nine Afrikaans speakers and 12 isiXhosa speakers. The duration of most recordings is between 5 and 10 seconds. Based on a brief inspection, the transcription texts of this dataset contain mostly full sentences and a few that contain short phrases.

As mentioned, we select recordings from the three datasets to create the final Afrikaans and isiXhosa datasets. Additionally, we use the Dutch and isiZulu data from the FLEURS dataset for our sequential fine-tuning experiments.

3.2.1. Approach for selecting recordings

To optimize training and GPU memory usage, we select recordings to maintain a roughly uniform distribution in terms of their durations. This approach mitigates the challenges of random batch selection during fine-tuning. Since batches are selected randomly, one batch may contain much longer recordings than another batch. Inconsistent batch durations may lead to suboptimal GPU memory allocation, leading to all of the GPU memory being used and triggering an exception. The NCHLT dataset contains much more data than the other two datasets. However, the NCHLT recordings are much shorter and of lower quality. Consequently, we omit a large portion of the NCHLT dataset in favor of

the recordings from the other two datasets. Figure 3.1 describes the histograms of the recording durations of the two datasets without removing outliers, Figure 3.2 describes the histograms of the recording durations of the two datasets after removing outliers. Note that we lose a significant portion of the data. However, it allows us to use larger batch sizes during fine-tuning.

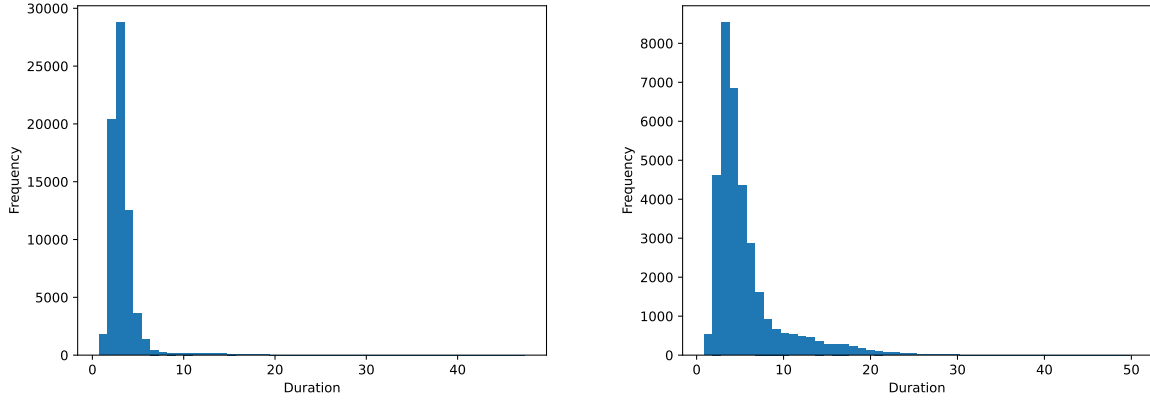


Figure 3.1: The histograms of the recording durations of the Afrikaans dataset (left) and the isiXhosa dataset (right) **without removing outliers**.

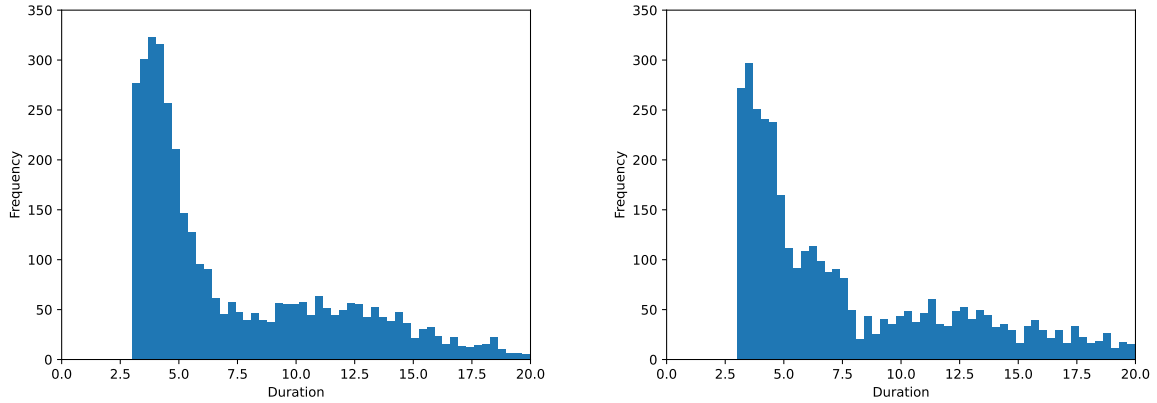


Figure 3.2: The histograms of the recording durations of the Afrikaans dataset (left) and the isiXhosa dataset (right) **after removing outliers**.

Both datasets contain approximately 7.5 hours of speech recordings (after removing outliers). Additionally, we ensure that the validation and test data does not contain recordings of speakers that appear in the training data (2.1.1). The transcription texts are pre-processed in the same way as the LM data, which is explained in the following section.

3.2.2. Language model data

The data used for training the LM (2.3.3) consists of Afrikaans and isiXhosa [Wikipedia dumps](#). The text is extracted from the dumps using a tool called the WikiExtractor [31]. The text is pre-processed by converting all characters to lowercase, removing all characters that are not used in the Afrikaans and isiXhosa language, and removing punctuation marks and other special characters.

3.3. Evaluation metric

Word error rate The word error rate (WER) [32] is equal to the number of character-level errors in the predicted transcript, divided by the number of words in the true transcript. One character-level error is corrected using one of three operations: inserting a new character, deleting an existing character, or substituting an existing character for a new character.

That concludes our experimental setup. Firstly, we discussed our main research question and how we will attempt to solve it using different experiments. Secondly, we explained our approach for collecting Afrikaans, Dutch, isiXhosa, and isiZulu speech data. Finally, we briefly discussed the training data for our LM, and we briefly explained the WER metric.

Chapter 4

Results

In this chapter, we discuss our experimental results. We first discuss the results of our basic fine-tuning experiments, and then we discuss the results of our sequential fine-tuning experiments. We do not include the results of all our models, only the best-performing Afrikaans and isiXhosa models for each training strategy. The results for all of our models are provided in Appendix A.

4.1. Basic fine-tuning results

For the basic fine-tuning experiments, each model is fine-tuned on one language. Since this study is focused on performing ASR for Afrikaans and isiXhosa, we only use Afrikaans and isiXhosa speech data for our basic fine-tuning experiments.

4.1.1. Afrikaans results

Table 4.1 provides the results of the best Afrikaans model using basic fine-tuning, with and without the use of a 5-gram LM. For more information about the hyperparameters and training history of this model, refer to the [model repository](#). Table A.1 in Appendix A provides the results of all our basic fine-tuning experiments for Afrikaans ASR.

Table 4.1: The WER of the best Afrikaans model using basic fine-tuning. The model is evaluated on the validation and test data of the Afrikaans dataset ([asr_af](#)).

| Model | WER | |
|---|------------|--------|
| | Validation | Test |
| Fine-tune on asr_af | 37.00% | 38.77% |
| Fine-tune on asr_af (with LM) | 26.67% | 28.30% |

4.1.2. isiXhosa results

Table 4.2 provides the results of the best isiXhosa model using basic fine-tuning, with and without the use of a 5-gram LM. For more information about the hyperparameters and training history of this model, refer to the [model repository](#). Table A.2 in Appendix A provides the results of all our basic fine-tuning experiments for isiXhosa ASR.

Table 4.2: The WER of the best isiXhosa model using basic fine-tuning. The model is evaluated on the validation and test data of the isiXhosa dataset ([asr_xh](#)).

| Model | WER | |
|---|------------|--------|
| | Validation | Test |
| Fine-tune on asr_xh | 50.08% | 50.52% |
| Fine-tune on asr_xh (with LM) | 40.14% | 41.47% |

4.2. Sequential fine-tuning results

For the basic fine-tuning experiments, each model is fine-tuned on two languages in two separate runs. We use the [FLEURS](#) dataset to perform basic fine-tuning experiments on Dutch (FLEURS_n1) and isiZulu (FLEURS_zu). Then, we use the best Dutch and isiZulu models for further fine-tuning on Afrikaans and isiXhosa respectively.

4.2.1. Afrikaans results

Table 4.3 provides the results of the best Afrikaans model using sequential fine-tuning, with and without the use of a 5-gram LM. For more information about the hyperparameters and training history of this model, refer to the [model repository](#). Table A.3 in Appendix A provides the results of all our sequential fine-tuning experiments for Afrikaans ASR.

Table 4.3: The WER of the best Afrikaans model using sequential fine-tuning. The model is evaluated on the validation and test data of the Afrikaans dataset ([asr_af](#)).

| Model | WER | |
|---|------------|--------|
| | Validation | Test |
| Fine-tune on FLEURS_n1 and asr_af | 35.17% | 37.49% |
| Fine-tune on FLEURS_n1 and asr_af (with LM) | 25.97% | 27.50% |

4.2.2. isiXhosa results

Table 4.4 provides the results of the best isiXhosa model using sequential fine-tuning, with and without the use of a 5-gram LM. For more information about the hyperparameters and training history of this model, refer to the [model repository](#). Table A.4 in Appendix A provides the results of all our sequential fine-tuning experiments for isiXhosa ASR.

Table 4.4: The WER of the best isiXhosa model using sequential fine-tuning. The model is evaluated on the validation and test data of the isiXhosa dataset ([asr_xh](#)).

| Model | WER | |
|---|------------|--------|
| | Validation | Test |
| Fine-tune on FLEURS_zu and asr_xh | 50.11% | 49.89% |
| Fine-tune on FLEURS_zu and asr_xh (with LM) | 40.11% | 40.51% |

4.3. Discussion of results

For our best Afrikaans models, with and without an LM (Table 4.1 and 4.3), sequential fine-tuning performed slightly better on the validation and test WER. For our best isiXhosa models, with and without an LM (Table 4.2 and 4.4), sequential fine-tuning also performed slightly better on the validation and test WER. Referring to Appendix A, several other models that used sequential fine-tuning (Table A.3 and A.4) performed better than most of the models that used basic fine-tuning (Table A.1 and A.2). Despite our results, the disadvantage of sequential fine-tuning is that it requires more computational resources since the strategy involves training a model twice. That concludes our experimental results and the main findings of this study.

Chapter 5

Conclusion

We investigated whether the use of additional speech data from Dutch and isiZulu can help develop ASR systems for Afrikaans and isiXhosa. We presented two training strategies. The first strategy involves training only on the target language, and the second strategy involves training on a related language first and then on the target language. Each model is trained by fine-tuning the XLS-R model for ASR. We train several models for Afrikaans and isiXhosa using the two strategies, and we evaluate the performance of each model on our validation and test data. Our experimental results show that the use of additional data from Dutch and isiXhosa slightly improves the performance of our Afrikaans and isiXhosa models, while requiring more computational resources to train our models. One major limitation of our work is the generalization ability of our models. This leads us to what we believe should be addressed in future work.

5.1. Future work

The Afrikaans and isiXhosa datasets were used for our experiments are very small (approximately 7.5 hours of speech data each). We recommend for future work that more labeled speech data for Afrikaans and isiXhosa should be collected and used for training and evaluating the ASR models. This is challenging because both Afrikaans and isiXhosa are currently considered under-resourced languages.

Additionally, we believe that, given enough computational resources, performing wav2vec 2.0 pre-training on a large dataset of unlabeled speech data may improve the accuracy of our Afrikaans and isiXhosa models. We believe that there are two ways this can be done. The first method involves training wav2vec 2.0 from scratch using randomly initialized weights, which would be computationally expensive. The second method involves training a wav2vec 2.0 model that is initialized with the same weights of the XLS-R model, which is basically fine-tuning XLS-R on unlabeled speech data using the wav2vec 2.0 objective function (2.2.4). We also recommend that different amounts of data for each language should be used in different experiments to see whether cross-lingual representations result in better accuracy (after fine-tuning) compared to mono-lingual representations.

Bibliography

- [1] M. Wald, “Using automatic speech recognition to enhance education for all students: Turning a vision into reality,” in *Frontiers in Education*, vol. 35. IEEE, 2005, pp. S3G–S3G. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/1612286>
- [2] N. Terbeh, M. Labidi, and M. Zrigui, “Automatic speech correction: A step to speech recognition for people with disabilities,” in *Fourth International Conference on Information and Communication Technology and Accessibility*. IEEE, 2013, pp. 1–6. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/6815303>
- [3] M. Wald, “Captioning for deaf and hard of hearing people by editing automatic speech recognition in real time,” in *International Conference on Computers for Handicapped Persons*. Springer, 2006, pp. 683–690. [Online]. Available: https://link.springer.com/chapter/10.1007/11788713_100
- [4] W. contributors, “Comparison of afrikaans and dutch,” Wikipedia, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Comparison_of_Afrikaans_and_Dutch
- [5] R. Lakey, “Progressively repurpose cutting-edge models,” MSS Cape Town, 2015. [Online]. Available: <https://msskapstadt.de/progressively-repurpose-cutting-edge-models/#:~:text=Both%20isiXhosa%20and%20isiZulu%20are,South%20Africa%20as%20a%20whole>
- [6] H. Kamper, F. De Wet, T. Hain, and T. Niesler, “Capitalising on north american speech resources for the development of a south african english large vocabulary speech recognition system,” *Computer Speech & Language*, vol. 28, no. 6, pp. 1255–1268, 2014. [Online]. Available: <https://www.sciencedirect.com/science/article/abs/pii/S0885230814000369>
- [7] L. Rabiner, “A tutorial on hidden markov models and selected applications in speech recognition,” *Proceedings of the IEEE*, vol. 77, pp. 257–286, 1989. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/18626>
- [8] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, “Listen, attend and spell: A neural network for large vocabulary conversational speech recognition,” in *International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2016, pp. 4960–4964. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/7472621>

- [9] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, “wav2vec 2.0: A framework for self-supervised learning of speech representations,” in *Advances in Neural Information Processing Systems*, vol. 33. Curran Associates, Inc., 2020, pp. 12 449–12 460. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/92d1e1eb1cd6f9fba3227870bb6d7f07-Paper.pdf
- [10] A. Babu, C. Wang, A. Tjandra, K. Lakhotia, Q. Xu, N. Goyal, K. Singh, P. von Platen, Y. Saraf, J. Pino, A. Baevski, A. Conneau, and M. Auli, “XLS-R: self-supervised cross-lingual speech representation learning at scale,” *Preprint*, 2021. [Online]. Available: <https://arxiv.org/abs/2111.09296>
- [11] W. Cochran, J. Cooley, D. Favin, H. Helms, R. Kaenel, W. Lang, G. Maling, D. Nelson, C. Rader, and P. Welch, “What is the fast fourier transform?” *Proceedings of the IEEE*, vol. 55, pp. 1664–1674, 1967. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/1447887>
- [12] J. W. Cooley, P. A. W. Lewis, and P. D. Welch, “The fast fourier transform and its applications,” *Transactions on Education*, vol. 12, pp. 27–34, 1969. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/4320436>
- [13] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Draft, 2023, vol. 3. [Online]. Available: <https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf>
- [14] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, “Librispeech: An asr corpus based on public domain audio books,” in *International Conference on Acoustics, Speech and Signal Processing*, vol. 1. IEEE, 2015, pp. 5206–5210. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/7178964>
- [15] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” *Preprint*, 2016. [Online]. Available: <https://arxiv.org/abs/1607.06450>
- [16] D. Hendrycks and K. Gimpel, “Bridging nonlinearities and stochastic regularizers with gaussian error linear units,” *Preprint*, 2016. [Online]. Available: <http://arxiv.org/abs/1606.08415>
- [17] J. Bgn, “An illustrated tour of wav2vec 2.0,” Blogpost, 2021. [Online]. Available: <https://jonathanbgn.com/2021/09/30/illustrated-wav2vec-2.html>
- [18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017, pp.

- 1–15. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf
- [19] J. Alammam. (2018) The illustrated transformer. Blogpost. [Online]. Available: <https://jalammar.github.io/illustrated-transformer/>
- [20] P. Shaw, J. Uszkoreit, and A. Vaswani, “Self-attention with relative position representations,” in *North American Chapter of the Association for Computational Linguistics*, vol. 1, 2018, pp. 464–468. [Online]. Available: <http://aclanthology.lst.uni-saarland.de/N18-2074.pdf>
- [21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems*, vol. 25. Curran Associates, Inc., 2012, pp. 1–9. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf
- [22] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, “Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks,” in *International Conference on Machine Learning*, vol. 12. Association for Computing Machinery, 2006, pp. 369–376. [Online]. Available: <https://doi.org/10.1145/1143844.1143891>
- [23] A. Hannun, “Sequence modeling with ctc,” Distill, 2017. [Online]. Available: <https://distill.pub/2017/ctc>
- [24] C. E. Shannon, “A mathematical theory of communication,” *The Bell System Technical Journal*, vol. 27, pp. 379–423, 1948. [Online]. Available: <http://plan9.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf>
- [25] H. Ney, U. Essen, and R. Kneser, “On structuring probabilistic dependences in stochastic language modelling,” *Computer Speech & Language*, vol. 8, pp. 1–38, 1994. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0885230884710011>
- [26] P. von Platen and other contributors, “Speech recognition pre-training,” GitHub, 2021. [Online]. Available: <https://github.com/huggingface/transformers/tree/main/examples/pytorch/speech-pretraining>
- [27] W. contributors, “Early stopping,” Wikipedia, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Early_stopping

- [28] E. Barnard, M. H. Davel, C. van Heerden, F. De Wet, and J. Badenhorst, “The nchlt speech corpus of the south african languages,” vol. 4, 2014, pp. 194–200. [Online]. Available: <https://pta-dspace-dmz.csir.co.za/dspace/handle/10204/7549>
- [29] A. Conneau, M. Ma, S. Khanuja, Y. Zhang, V. Axelrod, S. Dalmia, J. Riesa, C. Rivera, and A. Bapna, “Fleurs: Few-shot learning evaluation of universal representations of speech,” in *Spoken Language Technology Workshop*. IEEE, 2022, pp. 798–805. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/10023141>
- [30] D. van Niekerk, C. van Heerden, M. Davel, N. Kleynhans, O. Kjartansson, M. Jansche, and L. Ha, “Rapid development of tts corpora for four south african languages,” in *Interspeech*, vol. 1, 2017, pp. 2178–2182. [Online]. Available: <http://dx.doi.org/10.21437/Interspeech.2017-1139>
- [31] G. Attardi, “Wikiextractor,” GitHub, 2015. [Online]. Available: <https://github.com/attardi/wikiextractor>
- [32] W. contributors, “Word error rate,” Wikipedia, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Word_error_rate

Appendix A

Full experimental results

A.1. Basic fine-tuning results

A.1.1. Afrikaans results

Table A.1 provides the results of all our Afrikaans models using basic fine-tuning. Run 8 is our best model, and we also provide the repository for [Run 8 \(with LM\)](#).

Table A.1: The results of all our Afrikaans model using basic fine-tuning. The model is evaluated on the validation and test data of the Afrikaans dataset ([asr_af](#)).

| Model | Batch Size | Gradient Accumulation Steps | Evaluation Steps | Early Stopping Patience | WER (Val. Set) | WER (Test Set) |
|------------------------------|------------|-----------------------------|------------------|-------------------------|----------------|----------------|
| Run 1 | 4 | 3 | 50 | 3 | 42.46% | 43.31% |
| Run 2 | 4 | 3 | 100 | 3 | 42.47% | 43.42% |
| Run 3 | 4 | 3 | 250 | 2 | 43.06% | 43.66% |
| Run 4 | 4 | 3 | 400 | 2 | 37.89% | 38.41% |
| Run 5 | 8 | 2 | 50 | 3 | 37.98% | 38.01% |
| Run 6 | 8 | 2 | 100 | 3 | 41.33% | 42.49% |
| Run 7 | 8 | 2 | 250 | 2 | 41.83% | 42.12% |
| Run 8 | 8 | 2 | 400 | 2 | 37.00% | 38.77% |

A.1.2. isiXhosa results

Table A.2 provides the results of all our isiXhosa models using basic fine-tuning. Run 3 is our best model, and we also provide the repository for [Run 3 \(with LM\)](#).

Table A.2: The results of all our isiXhosa model using basic fine-tuning. The model is evaluated on the validation and test data of the isiXhosa dataset ([asr_xh](#)).

| Model | Batch Size | Gradient Accumulation Steps | Evaluation Steps | Early Stopping Patience | WER (Val. Set) | WER (Test Set) |
|------------------------------|------------|-----------------------------|------------------|-------------------------|----------------|----------------|
| Run 1 | 4 | 3 | 50 | 3 | 61.32% | 62.39% |
| Run 2 | 4 | 3 | 100 | 3 | 60.48% | 61.74% |
| Run 3 | 4 | 3 | 250 | 2 | 50.08% | 50.52% |
| Run 4 | 4 | 3 | 400 | 2 | 62.65% | 64.76% |
| Run 5 | 8 | 2 | 50 | 3 | 54.36% | 54.94% |
| Run 6 | 8 | 2 | 100 | 3 | 53.06% | 53.88% |
| Run 7 | 8 | 2 | 250 | 2 | 55.13% | 52.11% |
| Run 8 | 8 | 2 | 400 | 2 | 53.87% | 53.71% |

A.2. Sequential fine-tuning results

A.2.1. Afrikaans results

Table A.3 provides the results of all our Afrikaans models using sequential fine-tuning. Run 6 is our best model, and we also provide the repository for [Run 6 \(with LM\)](#).

Table A.3: The results of all our Afrikaans model using sequential fine-tuning. The model is evaluated on the validation and test data of the Afrikaans dataset ([asr_af](#)).

| Model | Batch Size | Gradient Accumulation Steps | Evaluation Steps | Early Stopping Patience | WER (Val. Set) | WER (Test Set) |
|------------------------------|------------|-----------------------------|------------------|-------------------------|----------------|----------------|
| Run 1 | 4 | 3 | 50 | 3 | 41.40% | 42.51% |
| Run 2 | 4 | 3 | 100 | 3 | 38.75% | 38.66% |
| Run 3 | 4 | 3 | 250 | 2 | 36.71% | 37.16% |
| Run 4 | 4 | 3 | 400 | 2 | 40.41% | 42.25% |
| Run 5 | 8 | 2 | 50 | 3 | 37.23% | 37.78% |
| Run 6 | 8 | 2 | 100 | 3 | 35.17% | 37.49% |
| Run 7 | 8 | 2 | 250 | 2 | 36.72% | 37.16% |
| Run 8 | 8 | 2 | 400 | 2 | 37.77% | 38.28% |

A.2.2. isiXhosa results

Table A.4 provides the results of all our isiXhosa models using sequential fine-tuning. Run 7 is our best model, and we also provide the repository for [Run 7 \(with LM\)](#).

Table A.4: The results of all our isiXhosa model using sequential fine-tuning. The model is evaluated on the validation and test data of the isiXhosa dataset ([asr_xh](#)).

| Model | Batch Size | Gradient Accumulation Steps | Evaluation Steps | Early Stopping Patience | WER (Val. Set) | WER (Test Set) |
|------------------------------|------------|-----------------------------|------------------|-------------------------|----------------|----------------|
| Run 1 | 4 | 3 | 50 | 3 | 64.04% | 64.40% |
| Run 2 | 4 | 3 | 100 | 3 | 51.15% | 51.32% |
| Run 3 | 4 | 3 | 250 | 2 | 52.06% | 53.18% |
| Run 4 | 4 | 3 | 400 | 2 | 54.00% | 55.18% |
| Run 5 | 8 | 2 | 50 | 3 | 55.78% | 57.66% |
| Run 6 | 8 | 2 | 100 | 3 | 52.22% | 55.30% |
| Run 7 | 8 | 2 | 250 | 2 | 50.11% | 49.89% |
| Run 8 | 8 | 2 | 400 | 2 | 53.64% | 56.86% |

A.3. Additional fine-tuning results

A.3.1. Dutch results

Table A.5 provides the results of our best four Dutch models using basic fine-tuning.

Table A.5: The results of our best four Dutch model using basic fine-tuning. The model is evaluated on the validation and test data of `FLEURS_n1`.

| Model | Batch Size | Gradient Accumulation Steps | Evaluation Steps | Early Stopping Patience | WER (Val. Set) | WER (Test Set) |
|-------|------------|-----------------------------|------------------|-------------------------|----------------|----------------|
| Run 1 | 4 | 3 | 250 | 2 | 40.36% | 40.73% |
| Run 2 | 8 | 2 | 50 | 3 | 38.05% | 37.20% |
| Run 3 | 4 | 3 | 100 | 3 | 40.48% | 40.89% |
| Run 4 | 8 | 2 | 100 | 3 | 42.94% | 43.74% |

A.3.2. isiZulu results

Table A.6 provides the results of our best four isiZulu models using basic fine-tuning.

Table A.6: The results of our best four isiZulu model using basic fine-tuning. The model is evaluated on the validation and test data of `FLEURS_zu`.

| Model | Batch Size | Gradient Accumulation Steps | Evaluation Steps | Early Stopping Patience | WER (Val. Set) | WER (Test Set) |
|-------|------------|-----------------------------|------------------|-------------------------|----------------|----------------|
| Run 1 | 4 | 3 | 250 | 3 | 61.41% | 60.23% |
| Run 2 | 8 | 2 | 100 | 3 | 66.93% | 66.09% |
| Run 3 | 4 | 3 | 50 | 3 | 57.19% | 57.27% |
| Run 4 | 4 | 3 | 100 | 3 | 67.79% | 67.97% |