Stellenbosch
UNIVERSITY
IYUNIVESITHI
UNIVERSITEIT

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.

Stellenbosch UNIVERSITY IYUNIVESITHI UNIVERSITEIT | forward together sonke siya phambili saam vorentoe

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 is 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 woordelikse 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 / *Student number*	[illegible] Handtekening / *Signature*
L. Meyer Voorletters en van / *Initials and surname*	**6 November 2023** Datum / *Date*

Abstract

Automatic speech recognition (ASR) involves identifying the spoken words for a given speech recording and returning the text of the spoken words. ASR is an important technology used across many domains such as enhancing educational systems [1], helping people with disabilities [2], and automatic captioning [3]. Several high-quality ASR systems for English are developed such as Siri, Alexa, and Google Assistant. However, creating high-quality ASR systems for low-resource languages remains a challenge. In this study we perform ASR for two low-resource languages: Afrikaans and isiXhosa.

Our approach involves fine-tuning XLS-R for ASR. XLS-R is a large-scale wav2vec 2.0 model that transforms speech recordings (audio data) into cross-lingual speech features. Two fine-tuning strategies are compared in this study, referred to as the *basic* fine-tuning strategy and the *sequential* fine-tuning strategy. Basic fine-tuning involves fine-tuning on one language, and sequential fine-tuning involves fine-tuning on two languages sequentially. For our sequential fine-tuning experiments, we fine-tune on Dutch before fine-tuning on Afrikaans, and we fine-tune on isiZulu before fine-tuning on isiXhosa. We create several ASR models using both fine-tuning strategies and our models are evaluated using the word error rate (WER) metric.

Our results conclude that the sequential fine-tuning strategy is more effective than the basic fine-tuning strategy for both Afrikaans and isiXhosa. By using sequential fine-tuning, our best Afrikaans model achieves a 0.3716 WER on our Afrikaans test set, and our best isiXhosa model achieves a 0.4989 WER on our isiXhosa test set. By using a seperately trained n-gram language model (LM) we achieve a 0.2796 WER for our best Afrikaans model, and a 0.3993 WER for our best isiXhosa model.

Chapter 1

Introduction

The task of automatic speech recognition (ASR) is to map a given speech recording to the text of the spoken words. The approach for performing ASR has remained the same for several years, which involves two main steps. In the first step the speech recording (audio data) is transformed into a feature representation known as speech features, and in the second step the speech features are mapped to a sequence of characters that represent the text. Traditionally, supervised learning techniques such as hidden Markov models have been used to perform ASR. However, self-supervised learning techniques have become popular within the field of ASR and other speech related tasks. Self-supervised learning allows us to learn feature representations using unlabeled data, which is particularly useful in low-resource settings. One of these techniques is wav2vec 2.0, which is machine learning model used to learn speech features. Wav2vec 2.0 can be fine-tuned to variety of speech-related tasks such as ASR, automatic speech translation, and speech classification. The issue is that training wav2vec 2.0 to learn speech features from unlabeled data is computationally expensive. Therefore, in this study we focus on fine-tuning pre-trained wav2vec 2.0 models for ASR for Afrikaans and isiXhosa.

We use the XLS-R model as our pre-trained model, which is a large-scale wav2vec 2.0 model trained on 128 different languages. Our Afrikaans and isiXhosa datasets are created by combining recordings from three different speech corpora to ensure speaker diversity. We compare two fine-tuning strategies: *basic* fine-tuning and *sequential* fine-tuning. Basic fine-tuning involves fine-tuning on one language, which is the typical strategy used for fine-tuning XLS-R. Sequential fine-tuning involves fine-tuning on two languages sequentially, specifically two related languages. Our assumption is that fine-tuning on a related language before fine-tuning on the target language may improve the accuracy of our ASR models. We use Dutch for our Afrikaans sequential fine-tuning experiments and we use isiZulu for our isiXhosa sequential fine-tuning experiments.

We create several models using both fine-tuning strategies, and all of the models are evaluated on our test sets using the word error rate (WER) metric. We found that the sequential fine-tuning approach resulted in more accurate Afrikaans and isiXhosa models overall, and our most accurate models were fine-tuned using the sequential fine-tuning strategy. However, the generalization ability of our best models are still limited, and this is most likely due to limited size of our datasets. 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 paper 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 features. We discuss how the connectionist temporal classification (CTC) algorithm is used to fine-tune speech features 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 recordings into a higher-dimensional feature representation called *speech features*. Then, in the second step we use supervised learning techniques to map speech features to a sequence of characters that are merged into a sentence.

Computing speech features is useful because recording data is difficult to interpret. Recording data, or audio data, consists of a sequence of floating point numbers called *samples*. The sequence of samples represent 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 audio data into speech features. The traditional approach is to compute speech features by transforming the audio data from the amplitude-time domain to the frequency-time domain using the fast Fourier transform (FFT) algorithm [4], [5].

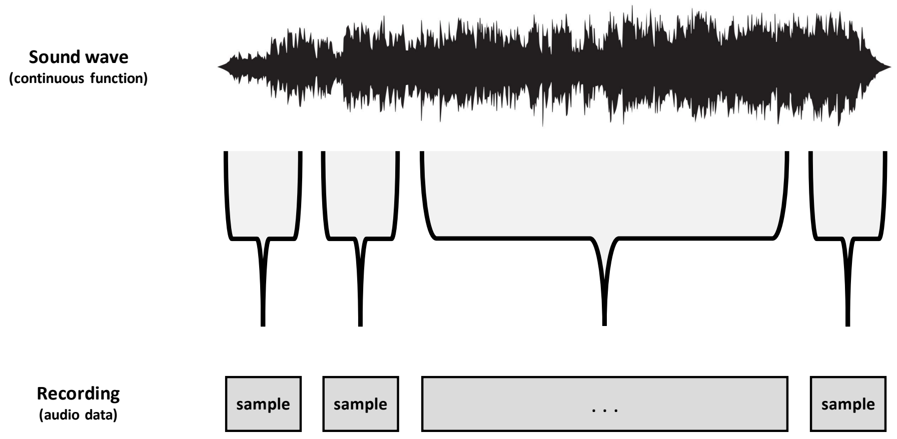

Figure 2.1: A diagram that explains what audio data represents. Note that this is an oversimplified explanation. The intent of the diagram is to explain why it is difficult to extract meaning directly from audio data.

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 audio 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 for ASR models.

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

The amount of training data. The training set should be as large as possible. A larger training set often leads to better accuracy and generalization performance for ASR models. A small dataset with few unique voices may lead to overfitting to the voices in the dataset.

Voice diversity. The training set should contain a diverse set 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 and conversational speech. The training set should contain both read speech and conversational speech. Speakers tend to speak more clearly when reading text from a transcript, in comparison to the speech of a conversation. Recent ASR models obtain very low error rates for recordings of read speech [6]. However, ASR for recordings of conversational speech is still a major challenge [6].

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 [7] is machine learning model used for learning speech features using *unlabeled*[1] audio 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 much 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 sets of the Librispeech [8].

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 audio 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.

[1]Unlabeled audio data simply refers to recordings without transcription text (labels).

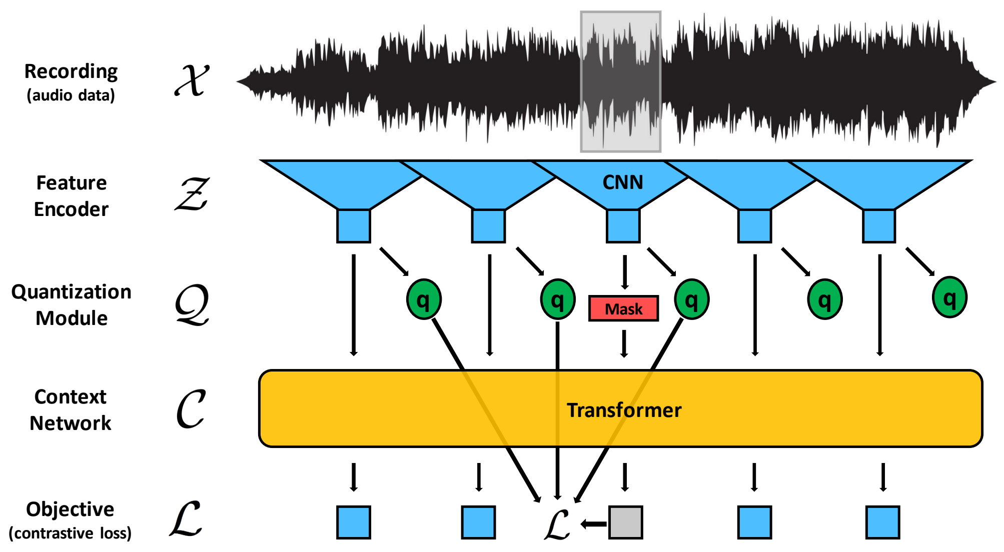

Figure 2.2: A diagram of the network architecture of wav2vec 2.0. This diagram is based on Figure 1 of the wav2vec 2.0 paper [7].

2.2.1. Feature encoder

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

The audio 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 [9], and the GELU [10] 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: $h : \mathcal{Z} \to \mathcal{Q}$. Unlike written language, which can be discretized into tokens such as characters or sub-words, speech does not have natural sub-units [11]. 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 $\mathrm{FF}_1 : \mathbb{R}^f \to \mathbb{R}^{G \times V}$ and $\mathrm{FF}_2 : \mathbb{R}^d \to \mathbb{R}^f$. In the second step above, the codebook entry $\mathbf{e}_g$ is chosen as the one with the argmax 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

$$p_{g,v} = \frac{\exp\left(\mathbf{l}^{(t)}_{g,v} + n_v\right)/\tau}{\sum\limits_{k=1}^{V} \exp\left(\mathbf{l}^{(t)}_{g,k} + n_k\right)/\tau}, \tag{2.1}$$

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 $i = \mathrm{argmax}_j p_{g,j}$, 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: $g : \mathcal{Z} \to \mathcal{C}$. The main component of the context network is a Transformer encoder [12]. Due to the popularity of Transformers, we omit the details of the Transformer architecture. However, we recommend illustrated guides such as [13] 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* [14] instead of absolute positional encoding. The relative positional encoding is implemented using grouped convolution [15].

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

The output after processing the latent feature vectorsaccording to the steps above are fed into the Transformer encoder, which results in a sequence of contextualized representations $\mathbf{c}^{(1)}, \dots \mathbf{c}^{(T)}$. The details for the Transformer encoder of 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 $\mathbf{z}_{\mathrm{M}}$, 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 $\mathbf{c}^{(t)}$. Additionaly, 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. A detailed explanation of the training objective follows 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

$$\mathcal{L}_m = -\log \frac{\exp(\mathrm{sim}(\mathbf{c_t}, \mathbf{q_t})\kappa)}{\sum_{\tilde{\mathbf{q}} \sim \mathcal{Q}_t} \exp(\mathrm{sim}(\mathbf{c_t}, \tilde{\mathbf{q}})\kappa)}, \tag{2.2}$$

where κ represents a constant temperature, and $\mathrm{sim}(a, b)$ 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:

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

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 vocabulary[2] 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) [16] 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 [6]. 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

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

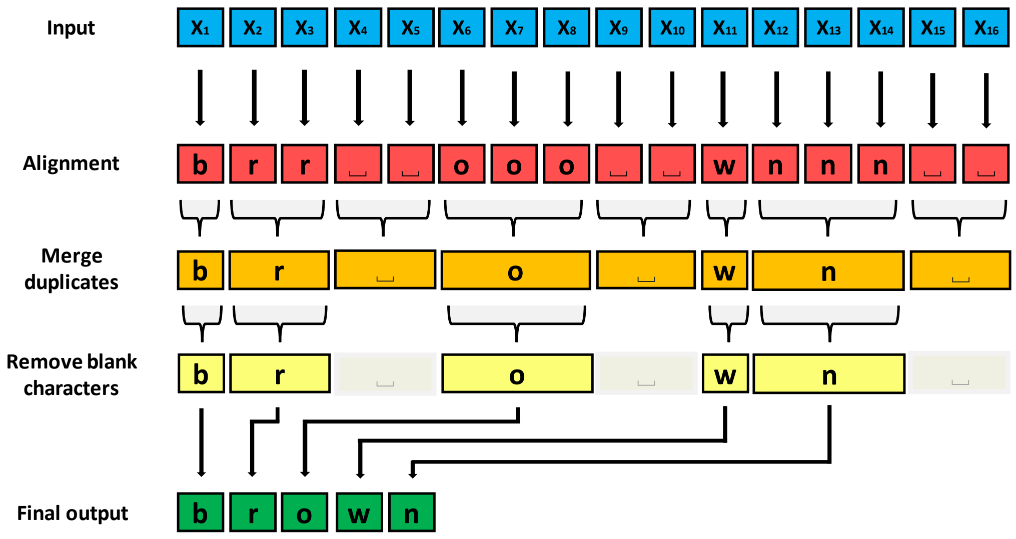

Figure 2.3: A diagram that describes the alignment procedure of CTC. This diagram is based on Figure 16.8 of [6].

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 [6], we define the collapsing function as a mapping $B : A \to Y$ 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).

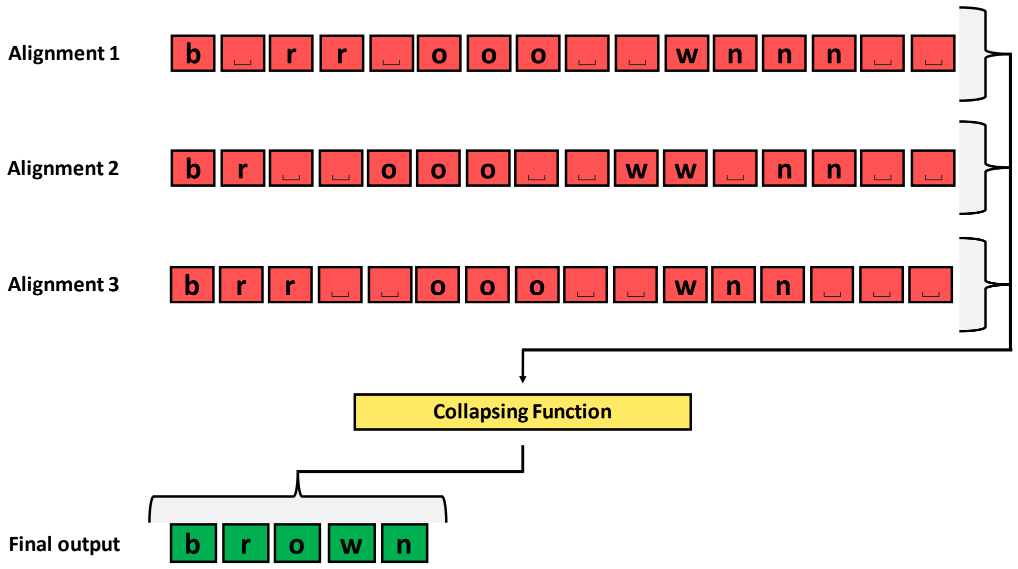
Alignment 1
b ␣ r r ␣ o o o ␣ ␣ w n n n ␣ ␣
Alignment 2
b r ␣ ␣ o o o ␣ ␣ w w ␣ n n ␣ ␣
Alignment 3
b r r ␣ ␣ o o o ␣ ␣ w n n ␣ ␣ ␣
Collapsing Function
Final output
b r o w n

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 [6].

In [6], 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 $X = \{x^{(1)}, \ldots, x^{(T)}\}$, denoted by $P_{\mathrm{CTC}}(Y|X)$. 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:

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

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 $a^{(t)} \in \{a^{(1)}, \ldots, a^{(T)}\}$ is computed independently of all the other alignment characters. Hence, we obtain the expression:

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

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 $\{a^{(1)}, \ldots, a^{(T)}\}$, 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 $\{\hat{a}^{(1)}, \ldots, \hat{a}^{(T)}\}$. The reason for this is that there are many possible alignments that lead to the same output text. Therefore, the most probable output text $\hat{Y}$, for a given X, corresponds to the maximum summation over the probabilities of all its possible alignments:

$$\begin{aligned} \hat{Y} = \mathrm{argmax}_Y P_{\mathrm{CTC}}(Y|X) &= \mathrm{argmax}_Y \left(\sum_{A \in B^{-1}(Y)} P(A|X) \right) && (2.6) \\ &= \mathrm{argmax}_Y \left(\sum_{A \in B^{-1}(Y)} \prod_{t=1}^{T} p(a^{(t)}|x^{(t)}) \right). && (2.7) \end{aligned}$$

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 [17]. 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 training objective function is the negative log-likelihood. Formally, the loss for a dataset D is equal to the summation of the negative log-likelihoods of the correct output Y for each corresponding input X:

$$\begin{aligned} L_{\text{CTC}} &= \sum_{(X,Y)\in D} -\log P_{\text{CTC}}(Y|X) && (2.8) \\ &= \sum_{(X,Y)\in D} -\log \left(\sum_{A\in B^{-1}(Y)} \prod_{t=1}^{T} p(a^{(t)}|x^{(t)}) \right). && (2.9) \end{aligned}$$

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*. A detailed explanation of the forward-backward algorithm used for training CTC is given in [17].

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 over the data. Therefore, the CTC algorithm commonly predicts sentences that contain obvious spelling mistakes. We can mitigate this issue by using a seperately trained language model (LM). This is done by adding additional terms in the CTC loss function with interpolation weights:

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

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 [18], [19]. We use 5-gram LMs, due to its popularity in recent ASR literature. We omit the details of n-grams and Kneser-Ney smoothing. However, we recommend [6] to interested readers.

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 unstable” (see NOTE 1 of [20]).

2. Performing one experiment of the large-scale wav2vec 2.0 model, using 8 GPU V100s with 16 GB RAM each, requires 7 days to finish training [20].

At the time of conducting this study, we did 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 model used in this study.

2.4.1. XLS-R

XLS-R [21] 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 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 $p_l \sim \left(\frac{n_l}{N}\right)^a$ where:

- $l \in \{1, \ldots, L\}$ 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 to select 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. The risk of not using this strategy is that some batches may contain recordings of only one language, which encourages the model to learn mono-lingual speech features and slows down training.

The total number of hours of all the unlabeled data combined is about 436k hours. The authors use data from 24 high-resource languages, 17 mid-resource languages, and 88 low-resource languages. Each high-resource language contains more than 1k hours of data, each mid-resource language contains more than 100 hours (but less than 1k hours) of data, and each low-resource language contains less than 100 hours of data.

Fine-tuning XLS-R results in better performance than the previous state-of-the-art for several benchmark datasets for every downstream task. This demonstrates the generalization ability of XLS-R. We are at the end of the background chapter. In the following chapter, we discuss our experimental setup and our fine-tuning experiments.

Chapter 3

Experimental Setup

This chapter provides our research questions, and how we attempt to answer these questions using the results of different experiments. We discuss the datasets used for our ASR models, as well as the 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 questions and experiments

3.1.1. Research questions

The focus of our research is determining how to obtain the best Afrikaans and isiXhosa ASR models with limited data and computational resources. Our approach involves fine-tuning (2.2.5) the XLS-R (300M) model using different fine-tuning strategies. We focus on two strategies in this study. The first (*basic*) strategy involves fine-tuning on one dataset (one language), which is the typical strategy for fine-tuning XLS-R (300M). The second strategy involves fine-tuning on two datasets (two languages) seperately. The second strategy is referred to as the *sequential* fine-tuning strategy. The intent of the sequential fine-tuning strategy is to first fine-tune on one language (e.g. Dutch), and then fine-tune on a different, but related, language (e.g. Afrikaans). We create several models using both strategies and each model is evaluated on a validation and test set using the word error rate (WER) (3.3). Finally, we choose the best model from both strategies for each language, add an LM (2.3.3) to each model to boost performance, and compare their final results.

3.1.2. Experiments

For the basic fine-tuning strategy, we fine-tune using either Afrikaans or isiXhosa speech data. For the sequential fine-tuning strategy, we fine-tune using either Dutch and then Afrikaans speech data, or isiZulu and then isiXhosa speech data. We chose Dutch and isiZulu, because Dutch is similar to Afrikaans [22] and isiZulu is similar to isiXhosa [23]. We first find the best Dutch and isiZulu model by using the basic fine-tuning strategy, and then we fine-tune those models to Afrikaans and isiXhosa respectively.

For each experiment, we change the values of the following hyperparameters: `batch size`, `gradient accumulation steps`, `evaluation steps`, and `patience`. The `evaluation steps` is the number of (gradient) steps between each evaluation on the validation set, and the `patience` is the parameter used for early stopping [24]. We use early stopping to prevent overfitting on the training data.

3.2. Data

We use three different speech corpora[1] to create an Afrikaans dataset and an isiXhosa dataset, which contains labeled speech recordings. We also use one of the corpora to create Dutch and isiZulu datasets (for sequential fine-tuning). We discuss the three corpora in the paragraphs below.

NCHLT. The NCHLT [25] corpus contains speech recordings 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 corpora, with most recordings being between 2 and 6 seconds. Based on brief inspection, the transcription texts of this dataset contain very few words, and rarely contain full sentences.

FLEURS. The FLEURS [26] corpus contains speech recordings for 102 different languages, including Afrikaans, Dutch, isiXhosa, and isiZulu. Each language has its own training, validation, and test set split. No information is given about the number of speakers for each language. The FLEURS recordings (on average) are the longest of the three corpora, with most recordings being between 7 and 20 seconds. Based on brief inspection, the transcription texts of this dataset contain full sentences.

High-quality TTS. The High-quality text-to-speech (TTS) [27] corpus contains high-quality transcribed audio 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 are between 5 and 10 seconds. Based on 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 corpora to create the final Afrikaans and isiXhosa datasets. Additionaly, we use the Dutch and isiZulu data from the FLEURS dataset for our sequential fine-tuning experiments.

[1]The word "corpora" is the plural form of the word "corpus".

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 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 h istograms 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 and as we mentioned the NCHLT recordings are low-quality recordings of short phrases not considered to be full sentences.

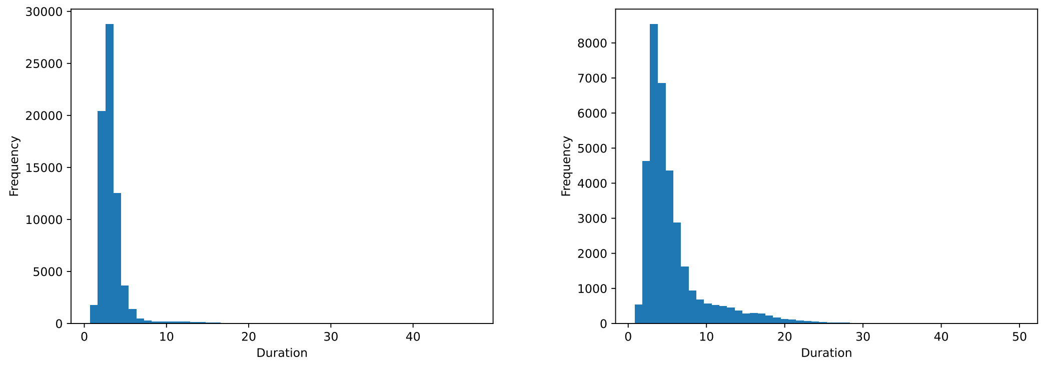

Figure 3.1: The histograms of the recording durations of the Afrikaans dataset (left) and the isiXhosa dataset (right) **without removing outliers**.

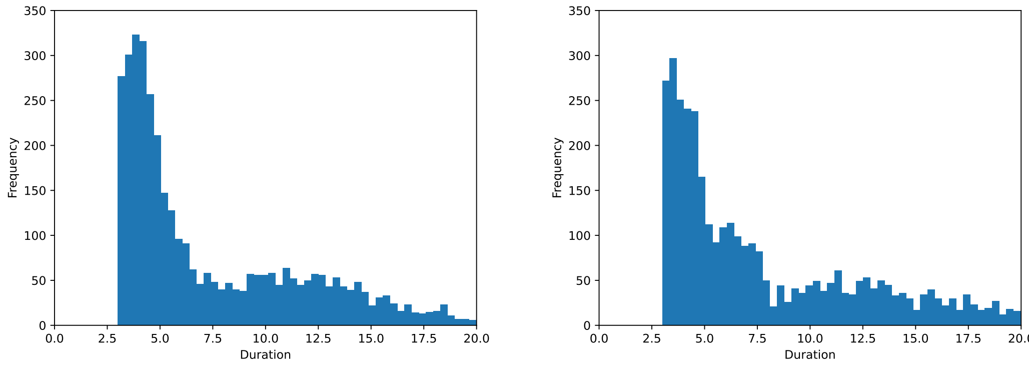

Figure 3.2: The histograms of the recording durations of the Afrikaans dataset (left) and the isiXhosa dataset (right) **after removing outliers**.

Both datasets contain approximately 7.5 hours of speech recordings (after removing outliers). Additionaly, we ensure that the validation and test sets do not contain recordings of speakers that appear in the training set (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 Wikipedia dumps. The text is extracted from the dumps using a tool called the WikiExtractor [28]. 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 metrics

Word error rate The word error rate (WER) [29] 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.

Chapter 4

Results

In this chapter, we discuss our experimental results. We first discuss the results for our basic fine-tuning experiments, and then we discuss the results for our sequential fine-tuning experiments. We do not include the results of all our experiments, only the experiments that resulted in the best WER (3.3). The results for all of our experiments 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 the basic fine-tuning strategy, with and without the use of a 5-gram LM (Kneser-Ney smoothing). 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 the basic fine-tuning strategy. The model is evaluated on the validation and test data of the Afrikaans dataset (`asr_af`).

	WER	
Model	Validation	Test
Fine-tune on `asr_af`	0.3798	0.3801
Fine-tune on `asr_af` (with LM)	0.2908	0.2874

4.1.2. isiXhosa results

Table 4.2 provides the results of the best isiXhosa model using the basic fine-tuning strategy, with and without the use of a 5-gram LM (Kneser-Ney smoothing). 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 the basic fine-tuning strategy. The model is evaluated on the validation and test data of the isiXhosa dataset (`asr_xh`).

	WER	
Model	Validation	Test
Fine-tune on `asr_xh`	0.5008	0.5052
Fine-tune on `asr_xh` (with LM)	0.4014	0.4147

4.2. Sequential fine-tuning results

For the basic fine-tuning experiments, each model is fine-tuned on two language in two seperate runs. We use the FLEURS dataset to perform basic fine-tuning experiments on Dutch (`FLEURS_nl`) 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 the sequential fine-tuning strategy, with and without the use of a 5-gram LM (Kneser-Ney smoothing). 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 the sequential fine-tuning strategy. The model is evaluated on the validation and test data of the Afrikaans dataset (`asr_af`).

	WER	
Model	Validation	Test
Fine-tune on `FLEURS_nl` and `asr_af`	0.3671	0.3716
Fine-tune on `FLEURS_nl` and `asr_af` (with LM)	0.2743	0.2796

4.2.2. isiXhosa results

Table 4.4 provides the results of the best isiXhosa model using the sequential fine-tuning strategy, with and without the use of a 5-gram LM (Kneser-Ney smoothing). 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 the sequential fine-tuning strategy. The model is evaluated on the validation and test data of the isiXhosa dataset (`asr_xh`).

	WER	
Model	Validation	Test
Fine-tune on `FLEURS_zu` and `asr_xh`	0.4945	0.4989
Fine-tune on `FLEURS_zu` and `asr_xh` (with LM)	0.3923	0.3993

4.3. Discussion of results

For our best Afrikaans models, with and without an LM (Table 4.1 and 4.3), the sequential fine-strategy 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), the sequential fine-strategy performed slightly better on the validation and test WER. The results of several other models (n Appendix A) that used the sequential fine-tuning strategy are better than the best basic fine-tuning results. The advantage of the basic fine-tuning strategy is that it is computationally less expensive compared to the sequential fine-tuning strategy which involves two seperate runs.

Chapter 5

Conclusion

In this study we performed automatic speech recognition (ASR) for Afrikaans and isiXhosa. We introduced wav2vec 2.0, which is a machine learning model used to transform audio data into feature representations for speech. We explained how using wav2vec 2.0 for unsupervised pre-training is challenging given our computational resources, and proposed a feasible approach in which we instead fine-tune a wav2vec 2.0 model that has already been pre-trained. We chose to fine-tune XLS-R, a large-scale wav2vec 2.0 model that has been pre-trained on 128 languages in order to learn *cross-lingual* representations. We briefly discussed our datasets and our selection approach which mitigates some of the common issues of ASR. We performed multiple fine-tuning experiments using different hyperparameters and fine-tuning strategies. In particular, we compared a basic fine-tuning strategy, in which one language is used for fine-tuning, to a sequential fine-tuning strategy, in which one language is used to fine-tune the model and then the model is fine-tuned again using a different language. For the sequential fine-tuning experiments, we first fine-tuned on Dutch for our Afrikaans models, and we first fine-tuned on isiZulu for isiXhosa models. Our results conclude that the sequential fine-tuning strategy is more effective than the basic fine-tuning strategy for both Afrikaans and isiXhosa. By using sequential fine-tuning, our best Afrikaans model achieves a 0.3716 WER on our Afrikaans test set, and our best isiXhosa model achieves a 0.4989 WER on our isiXhosa test set. By using a seperately trained n-gram language model (LM) we achieve a 0.2796 WER for our best Afrikaans model, and a 0.3993 WER for our best isiXhosa model. 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). This is because Afrikaans and isiXhosa are low-resource languages. 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.

Additionaly, we believe that performing wav2vec 2.0 pre-training using a large corpus 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. The second method involves training wav2vec 2.0 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 (Afrikaans and isiXhosa) representations results 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 *Proceedings Frontiers in Education 35th Annual Conference.* IEEE, 2005, pp. S3G–S3G.

[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 (ICTA).* IEEE, 2013, pp. 1–6.

[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.

[4] W. T. Cochran, J. W. Cooley, D. L. Favin, H. D. Helms, R. A. Kaenel, W. W. Lang, G. C. Maling, D. E. Nelson, C. M. Rader, and P. D. Welch, “What is the fast fourier transform?” *Proceedings of the IEEE*, vol. 55, no. 10, pp. 1664–1674, 1967.

[5] J. W. Cooley, P. A. Lewis, and P. D. Welch, “The fast fourier transform and its applications,” *IEEE Transactions on Education*, vol. 12, no. 1, pp. 27–34, 1969.

[6] D. Jurafsky and J. H. Martin, “Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition.”

[7] A. Baevski, H. Zhou, A. Mohamed, and M. Auli, “wav2vec 2.0: A framework for self-supervised learning of speech representations,” 2020.

[8] 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).* IEEE, 2015, pp. 5206–5210.

[9] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” *arXiv preprint arXiv:1607.06450*, 2016.

[10] D. Hendrycks and K. Gimpel, “Gaussian error linear units (gelus),” *arXiv preprint arXiv:1606.08415*, 2016.

[11] J. Bgn. (2021) An illustrated tour of wav2vec 2.0. [Online]. Available: https://jonathanbgn.com/2021/09/30/illustrated-wav2vec-2.html

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[13] J. Alammar. (2018) The illustrated transformer. [Online]. Available: https://jalammar.github.io/illustrated-transformer/

[14] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-attention with relative position representations," *CoRR*, vol. abs/1803.02155, 2018. [Online]. Available: http://arxiv.org/abs/1803.02155

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[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*, 2006, pp. 369–376.

[17] A. Hannun, "Sequence modeling with ctc," *Distill*, 2017, https://distill.pub/2017/ctc.

[18] 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

[19] H. Ney, U. Essen, and R. Kneser, "On structuring probabilistic dependences in stochastic language modelling," *Computer Speech & Language*, vol. 8, no. 1, pp. 1–38, 1994.

[20] P. von Platen and other contributors. (2021) Speech recognition pre-training. [Online]. Available: https://github.com/huggingface/transformers/tree/main/examples/pytorch/speech-pretraining

[21] 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," *arXiv preprint arXiv:2111.09296*, 2021.

[22] W. contributors, "Comparison of afrikaans and dutch," Wikipedia, 2023, accessed: October 27, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Comparison_of_Afrikaans_and_Dutch

[23] R. Lakey, “Progressively repurpose cutting-edge models,” MSS Cape Town, 2023, accessed: October 27, 2023. [Online]. Available: https://msskapstadt.de/progressively-repurpose-cutting-edge-models/#:~:text=Both%20isiXhosa%20and%20isiZulu%20are,South%20Africa%20as%20a%20whole

[24] W. contributors, “Early stopping,” Wikipedia, 2023, accessed: October 27, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Early_stopping

[25] E. Barnard, M. H. Davel, C. van Heerden, F. De Wet, and J. Badenhorst, “The nchlt speech corpus of the south african languages.” Workshop Spoken Language Technologies for Under-resourced Languages (SLTU), 2014.

[26] 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,” *arXiv preprint arXiv:2205.12446*, 2022. [Online]. Available: https://arxiv.org/abs/2205.12446

[27] 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 *Proc. Interspeech 2017*, 2017, pp. 2178–2182. [Online]. Available: http://dx.doi.org/10.21437/Interspeech.2017-1139

[28] G. Attardi, “Wikiextractor,” https://github.com/attardi/wikiextractor, 2015.

[29] W. contributors, “Word error rate,” Wikipedia, 2023, accessed: 2023-10-31. [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 the basic fine-tuning strategy. Note: Going to perform 4 more runs (af and xh) using basic fine-tuning.

Table A.1: The results of all our Afrikaans model using the basic fine-tuning strategy. 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	400	2	0.3879	0.3841
Run 2	8	2	50	3	0.3798	0.3801
Run 3	4	3	250	3	0.4306	0.4366
Run 4	8	2	100	3	0.4133	0.4229

A.1.2. isiXhosa results

Table A.2 provides the results of all our isiXhosa models using the basic fine-tuning strategy.

Table A.2: The results of all our isiXhosa model using the basic fine-tuning strategy. 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	250	2	0.5008	0.5052
Run 2	8	2	50	3	0.5436	0.5494
Run 3	8	2	100	3	0.5306	0.5388
Run 4	16	2	100	2	0.5008	0.5104

A.2. Sequential fine-tuning results

A.2.1. Afrikaans results

Table A.3 provides the results of all our Afrikaans models using the sequential fine-tuning strategy. Note[1]

Table A.3: The results of all our Afrikaans model using the sequential fine-tuning strategy. 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	0.3723	0.3778
Run 2	4	3	100	3	0.3875	0.3866
Run 3	4	3	250	2	0.3671	0.3716
Run 4	4	3	400	2	0.4041	0.4225
Run 5	8	2	50	3	0.4140	0.4251
Run 6	8	2	100	3	0.3517	0.3749
Run 7	8	2	250	2	0.3672	0.3716

A.2.2. isiXhosa results

Table A.4 provides the results of all our isiXhosa models using the sequential fine-tuning strategy.

Table A.4: The results of all our isiXhosa model using the sequential fine-tuning strategy. 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	0.6404	0.6440
Run 2	4	3	100	3	0.5115	0.5132
Run 3	4	3	250	2	0.5206	0.5318
Run 4	4	3	400	2	0.5400	0.5518
Run 5	8	2	50	3	0.5578	0.5766
Run 6	8	2	100	3	0.5222	0.5530
Run 7	8	2	250	2	0.4945	0.4989

[1]The results you seen on the model card do not match with the tabular results. I evaluate the models seperately using `src/test.ipynb`. I also need to re-run Run 8 (af and xh).

A.3. Additional fine-tuning results

A.3.1. Dutch results

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

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

Model	Batch Size	Gradient Accumulation Steps	Evaluation Steps	Early Stopping Patience	**WER (Val. Set)**	**WER (Test Set)**
Run 1	4	3	250	3	0.4036	0.4073
Run 2	8	2	100	3	0.3805	0.3720
Run 3	4	3	100	3	0.4048	0.4089
Run 4	16	2	100	2	0.4294	0.4374

A.3.2. isiZulu results

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

Table A.6: The results of all our isiZulu model using the basic fine-tuning strategy. 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	0.6141	0.6023
Run 2	8	2	100	3	0.6693	0.6609
Run 3	4	3	50	3	0.5719	0.5727
Run 4	4	3	100	3	0.6779	0.6797