Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Transfer Learning for low-resource languages

Ajchan Mamedov

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science

Supervisor: Dr. Anthony Ventresque

April 2024

Transfer Learning for low-resource languages

Ajchan Mamedov, Master of Science in Computer Science

University of Dublin, Trinity College, 2024

Supervisor: Dr. Anthony Ventresque

With the advancement of Automatic Speech Recognition (ASR) systems, a significant number of large ASR models trained on high-resource languages were created. We want to explore the application of transfer learning to develop ASR systems for low-resource languages, in our case on Lithuanian. Understanding the challenge of having a limited amount of labelled data available for low-resource languages, we want to speed up the fine-tuning of pre-trained models by making use of the common linguistic features across languages. The first experiment conducted uses clustering techniques to analyse feature similarity in cluster visualisations and silhouette scores. The second experiment performs fine-tuning on the pre-trained models and then gets predictions using the test dataset, to calculate Word Error Rate (WER) and Character Error Rate (CER) scores. Then we examine the results from experiment one and experiment two, to find a correlation. The results indicate that we can reduce the computational expenses of choosing models by pinpointing model compatibility prior to fine-tuning. From this, we can conclude that transfer learning can significantly reduce the need for large labelled datasets in building ASR systems for low-resource languages, like Lithuanian.

Acknowledgments

I would like to express my gratitude to my supervisors: Dr. Anthony Ventresque and Dr. Ellen Rushe, for guiding and supporting me throughout this dissertation process. I would also like to thank my family and friends for supporting me through the years of my college journey.

Ajchan Mamedov

University of Dublin, Trinity College
April 2024

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Automatic Speech Recognition (ASR) systems allow computers to understand spoken language and convert it into text. ASR systems are used in a lot of everyday places, such as transcription services, voice-activated commands and assistants that can be found in smartphones and smart devices. While these technologies have made significant advancements in accuracy and reliability, they are mostly available to languages with large amounts of labelled data.

1.1 Project

Developing ASR systems for low-resource languages is not an easy task, the main reason being the lack of labelled data available on these languages. With the recent advancements in deep learning, the newer ASR models require an extraordinary amount of data to create large models. Because of this, advancements in ASR technologies mainly benefited high-resource languages like English. I want to investigate if transfer learning techniques can enhance ASR systems for the Lithuanian language with a limited amount of labelled speech data. The essence of this project is using models that have been pre-trained on high-resource languages and then fine-tuning them using the low-resource Lithuanian language to get prediction results. By doing so I want to understand the viability of using transfer learning to create effective ASR systems for low-resource languages like Lithuanian.

1.2 Motivation

The motivation for this research comes from the unfairness of Automatic Speech Recognition technologies, where languages with a higher number of resources get more techno-

logical support than others. As a native speaker of several low-resource languages one of which is Lithuanian, I understand the benefits of having access to good ASR technology. A lot of the smartphones and smart devices in general that use speech recognition, don't have support for low-resource languages, such as Lithuanian. This limits the application of speech recognition technologies around the world and obstructs people from accessing certain digital services. This work is also motivated by the desire to use and preserve low-resource languages. As a lot of the smaller or indigenous languages are being used less and less, they are in danger of disappearing. Making ASR technologies available for these languages can keep them more active and relevant, as local people can use these technologies in their native languages and thus preserve them.

1.3 Aims

The main objective of this project is to understand the viability of using transfer learning to create effective ASR systems for low-resource languages like Lithuanian. Based on this we want to find out, if we can find the best pre-trained models for transfer learning, without needing to fine-tune them on Lithuanian by using less computationally intensive methods, like clustering. We want to know if we can cluster the features and see if there is a discernible structure. We then want to know if the fine-tuned model performance matches our cluster performance measure. If there is a correlation between the results, we can use clustering in the future to choose pre-trained models that best fit our target low-resource language. These would reduce the computational cost of training ASR systems using transfer learning for low-resource languages, with a limited amount of labelled data.

1.4 Structure

Chapter 2, State of the Art: provides a detailed overview of research in the field of ASRs. It starts by giving an introduction to the field and then talks about the history of ASR technologies. After that, it discusses the ethics of building large models from unsupervised models and then talks about several approaches for transfer learning, unsupervised learning and meta-learning.

Chapter 3, Design: provides a detailed description of the technologies that can be used, then formulates the problem. Then describe the challenges I identified and propose work to alleviate these challenges and design our software system. Lastly, it gives an overview of the Design with all its steps.

Chapter 4, Implementation: discusses the step by step process of the implementation. Explains each step in detail: data pre-processing, feature extraction and analysis, and fine-tuning.

Chapter 5, Evaluation: describes the experimental setup and then discusses the results I obtained from both experiments. Lastly, this section discusses how the results from both experiments correlate.

Chapter 6, Conclusions and Future Work: summarises the work I have done and the results obtained. It also explains several areas of possible future work.

Chapter 2

State of the Art

2.1 Background

Automatic speech recognition (ASR) is a technology that allows computers to interpret human speech and convert it into text. Early adoptions use algorithms to analyse the audio signals and decode them into text using engineered audio features as opposed to extracting these features automatically from speech data. Modern speech recognition systems use advanced machine learning techniques such as deep neural networks and transformers to improve the performance, accuracy and adaptability of ASRs. This technology is used in different applications, such as voice assistants, transcription services, and interactive voice response systems. It is fascinating to see how much the ASR technology has changed over the years.

2.1.1 History of automated speech recognition

Automatic speech recognition has a long history starting as early as the 1950s. The first attempt at speech recognition was made by Bell Laboratories who designed the 'Audrey' system, which could recognise single digits spoken with 90% accuracy (Davis, 1952). Since then the field of speech recognition has evolved together with technological advancements. Later on in the 1980s and 1990s Statistical methods were used for speech recognition, where the introduction of Hidden Markov Models(HMM) contributed to the development of modelling of time series data, like audio signals Rabiner (1989). I won't be delving too deeply into these methods as my project mainly focuses on deep learning-based ASR models.

2.1.2 History of AI automated speech recognition

The history of speech recognition started with the pivotal advancement that was made in the early 2010s when Deep Neural Network (DNN) methods were introduced to mainstream speech recognition (Hinton et al., 2012). These advancements led to the development of end-to-end deep learning models, such as Transformers Vaswani et al. (2017) originally made for translation purposes, which were then adapted to ASR by Dong et al. (2018). This led to an improvement in the performance, accuracy and reliability of automatic speech recognition models.

Hinton et al. (2012) combined the findings from several research groups that showed the effectiveness of DNNs in speech recognition. Before this publication, the main approach for speech recognition was using Gaussian Mixture Models(GMMs) in combination with HMMs (Rabiner, 1989). Although these models could handle the statistical aspects of speech recognition, they weren't able to model complex nonlinear relationships in acoustic signals very well. The authors chose to use DNNs with nonlinear activation functions as this allowed models to learn high-level abstractions contained in the data. Researchers back in the time of making these models faced the challenge of training deep networks, which suffered from vanishing gradients (Bengio et al., 1994). To address this challenge, pre-trained methods were used, such as Restricted Boltzmann Machines, to initialise the weights of the network Hinton and Salakhutdinov (2006). This meant that before fine-tuning the model on the labelled data, the network would be pre-trained in an unsupervised manner. This allowed authors to significantly improve the DNNs' performance. Another way the performance of DNNs was improved was by using GPUs to speed up the training process. Based on the above, we could argue that the DNN models were the first step in the development of modern AI ASRs, we could also say that the DNNs such as those developed Hinton et al. (2012) walked, so that the more complex models made after could run.

Throughout the years new ASR models were implemented with variations of existing Deep Neural Networks, such as Recurrent Neural Networks (RNNs). Following the development of more recent transformer models (Devlin et al., 2018), Dong et al. (2018) proposed the Speech-Transformer model, which uses the attention mechanism, which was originally designed for text processing and translation Vaswani et al. (2017), removing the need for recurrence. It was then adapted for speech recognition in this paper. The core innovation of this model is the use of self-attention, where representations of a sequence are computed by relating different positions of the input to each other. This allows for higher parallelisation of computations compared to RNNs and reduces the training times. This,

in turn, makes it easier to run the Speech-Transformer model on the GPUs and makes better use of the many computational cores the GPUs possess, increasing the speed of training and the performance of the model.

2.1.3 Ethics behind the large AI models

The evolution of ASR marks a significant shift from traditional Heuristic models to advanced neural Network-based models. From the work of Bender et al. (2021) we can notice that recently we have seen a massive push towards building Large Language Models (LLMs), such as BERT, GPT, and Switch-C, which have significantly improved the performance of Natural Language Processing (NLP) tasks. Although LLMs have shown significant performance improvements, it is still questionable whether there is a necessity to continually increase model sizes and if it is sustainable to do so given the associated risks and costs. The significant financial costs and environmental impact of training and deploying LLMs and large ASR models raise huge concerns. The name of the paper 'Stochastic Parrots,' refers to the idea that language and speech recognition models, regardless of their complexity or size, mimic or 'parrot human' language without understanding meaning or context. This mimicry is based on patterns found in massive text and speech corpora used to train these models. With this behaviour comes the risks of using vast, unlabelled/unvetted datasets scraped from the internet to train LLMs or ASR models. These datasets can often be biased and contain predominant viewpoints that can be harmful towards marginalised groups. Therefore, it is important to carefully select datasets and document any training material to ensure transparency while creating LLMs and ASR models. The authors state that having biases in the training data of ASRs can lead to biased outputs, where the model can perform better with certain accents and dialects than others. This could provide systematic disadvantages to individuals based on their speech patterns. There have been works to address these kinds of issues in ASR, one of which is (Lonergan et al., 2023).

Using Balanced Training Corpora for low resource languages for ASRs

In the paper by Lonergan et al. (2023), the authors explore the effectiveness of using balanced training corpora to enhance speech recognition models across different dialects of Irish (a low-resource language). To do this, the authors created 12 different datasets with varying numbers of corpora for each dialect and then trained the XLS-R wav2vec 2.0 (Schneider et al., 2019) pre-trained model with Connectionist Temporal Classification (CTC) Graves and Graves (2012). They initially trained the model with a corpora that was balanced across the three Irish dialects. Following this, these resulting models

were fine-tuned by the 12 datasets with varying dialect-specific material. The research showed that when using balanced corpora, there were significant differences in ASR performance across the dialects. The authors pointed out how the Ulster dialect consistently under-performed when compared to Munster and Connacht dialects. This suggests that just balancing the corpora is not enough in balancing an ASR system, as the quantity of data doesn't always translate to balanced performance. The authors note that different datasets with added dialect-specific data do improve recognition performance, but the improvement varies from dialect to dialect. All in all, the authors conclude that even though the usage of balanced corpora for low-resource languages does help with the recognition of dialect-specific data, it is not a panacea.

2.2 Related Work

In this section I will be talking about related work to what I am trying to accomplish, specifically approaches to low-resource ASRs.

2.2.1 Speech Recognition for Languages without Audio using n-gram statistics

The work done by Li et al. (2022a) investigates if developing speech recognition models for languages with little to no audio datasets is feasible, by relying on textual data or n-gram statistics. The authors hypothesised that they accomplished an effective speech recognition solution without direct audio. They planned to achieve this by using multilingual models, available dictionaries and rules of the languages, and lastly, a language model built from text data. To make this speech recognition model authors created a pipeline composed of three key elements: an acoustic model, a pronunciation model and a language model. The acoustic model utilises multilingual models to recognise phonemes of the target language even if they are not in the target language audio data. It does this by using an allophone-based multilingual architecture. This architecture makes use of a language-independent universal phone recognition model to identify the physical-level phone units from speech audio. The universal model is trained on high-resource languages like English and Mandarin. The authors make use of allophones, which are language-dependent phone to phoneme mappings which were encoded by phonologists Mortensen et al. (2020). These allophones allow the model to adapt to new languages without the need for direct training data from those languages. The Pronunciation model uses a multilingual grapheme-to-phoneme(G2P) model to predict phoneme pronunciations from text. If the languages happen to be high-resource and have existing dictionaries and rules, this model can be

trained directly by using this data. For languages with little to none of this data available, zero-shot Li et al. (2022b) or transfer learning is used, where the multilingual G2P model approximates pronunciation by using models from linguistically similar languages. It does this by selecting the top-k nearest languages and combining their models to predict the most likely phoneme sequence for the target language. The authors conducted experiments on 129 languages using two major datasets: Common Voice and CMU Wilderness. This speech recognition model achieves a respectable Character Error Rate (CER) and Word Error Rate (WER), where the CER and WER get better with a higher number of utterances(raw text) used, where when ten thousand utterances are used CER equals to 50% and 45% and WER equals to 79% and 69% in Common Voice Ardila et al. (2019) and Wilderness Black (2019) datasets respectively. The authors also pointed out that the performance varied significantly based on the linguistic characteristics of the target language. This is a new approach to ASR systems, but it is very complex in its structure as well as computationally expensive because multiple datasets need to be used and processed.

2.2.2 Pre-training on High-Resource ASR task to Improve Low-Resource Speech-to-Text Translation

Low-resource languages often don't have effective Speech-to-text Translation(ST) models, as they often lack training data. Usually, these systems would be built with a combination of ASR and Machine Translation(MT). However, in the case of low-resource languages, this method is not feasible. So paper by Bansal et al. (2018) tries to assess if pre-training an ST model on a high-resource ASR task can fix the issues of low-resource languages in ST. The authors chose to use the encoder-decoder model with attention 2.1, used for both of their ASR and ST models. The encoder processes the input data and converts it into a set of feature representations, while the decoder generates corresponding text in the target language. As we can see in the figure, the encoder consists of CNNs followed by bidirectional Long Short-term Memory Networks (LSTMs). The decoder makes use of an attention mechanism and its fully connected layers to focus on different parts of the speech input and generate text. The authors first want to pre-train this model on a high-resource ASR task, therefore they want to train the encoder-decoder model with attention on a high-resource language like English or French. Then they fine-tune the pre-trained ASR model on the low-resource ST task, where a smaller target dataset pairs audio from the low-resource language with text in the target language.

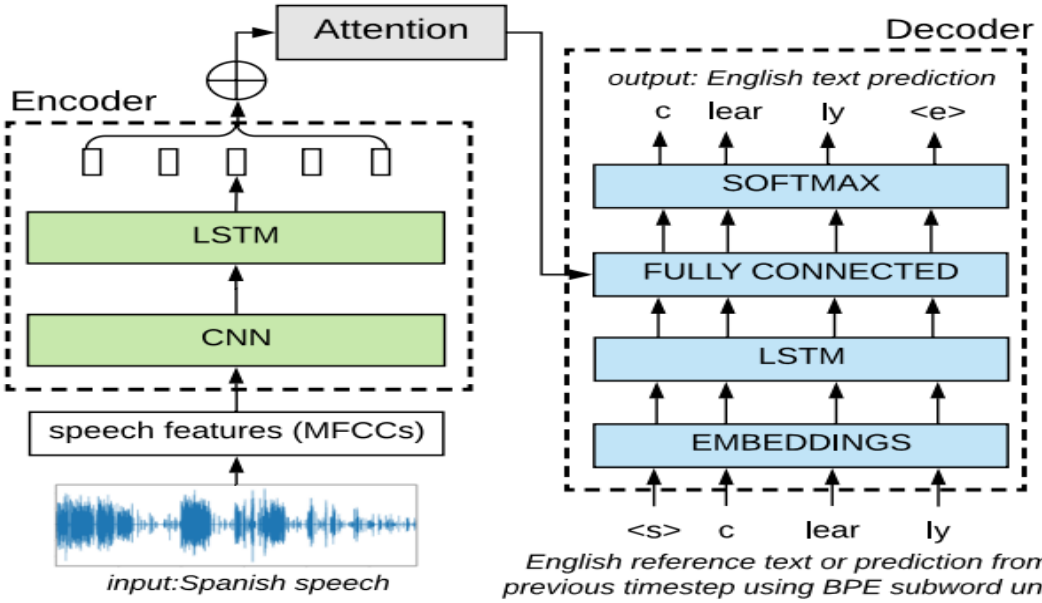


Figure 2.1: Encoder-decoder model with attention for ASR and ST (Bansal et al. (2018))

To test out how this methodology worked, the authors used two datasets for pre-training ASRs and two more for fine-tuning STs. To pre-train the English ASR they used The Switchboard Telephone Speech Corpus Godfrey and Holliman (1993) and to pre-train French ASR they used The GlobalPhone French Speech Corpus Schultz (2002). To fine-tune Spanish-English ST the authors used The Fisher Spanish Speech Corpus Graff et al. (2010) and to fine-tune Mboshi-French ST they used a corpus that includes about four hours of Mboshi speech translated into French Godard et al. (2017). The results show significant improvements when models are pre-trained on ASR tasks and then are fine-tuned for ST tasks. We can see this from the Spanish-English ST model result, where it achieves a BLEU Papineni et al. (2002) score improvement from 10.8 to 20.2. This happens when it first gets pre-trained on 300 hours of English ASR data and then fine-tuned on 20 hours of Spanish-English ST data. The authors conclude that this method is effective in training STs for low-resource languages by using high-resource ASR data to pre-train the models. It is also important to note that, even when trained on different languages compared to the target ST language, pre-training can provide a substantial performance boost to the model. This is largely due to the transferable nature of the learned acoustic models, which capture phonetic variations that are broadly applicable across languages.

2.2.3 Unsupervised Learning for Speech Recognition

Unsupervised learning can be dangerous due to the biases it can hold, but it still is a good tool for languages with scarce labelled data. Schneider et al. (2019) discusses the challenges of developing speech recognition models that require a lot of transcribed audio data for high performance. Due to the recent success of unsupervised training in Computer Vision and NLP tasks at the time of publishing of their paper, the authors proposed unsupervised pre-training as a solution for ASR models. They introduced the Wav2Vec model, which is designed to utilise large amounts of unlabelled audio data to learn general audio representations. The representations learned by wav2vec are then used to improve the training of speech recognition models, achieving significant improvements in WER even when very little transcribed data is available. The Wav2Vec model consists of 2 steps: a feature encoding step and a context aggregation step. The raw audio input is passed through an encoder network, which is a convolution neural network (CNN), that consists of five convolutional layers, where each layer has a progressively more abstract representation of the input audio. This encoder finds and saves local acoustic features from the raw audio without requiring any information that is labelled/transcribed. After the encoding step, a context network consisting of nine convolutional layers aggregates these features over time to create a temporal context. The context network also has a receptive field of 210ms, allowing it to process 210ms of audio at once, enhancing the performance wav2vec model. To test the performance of the wav2vec model the authors used Wall Street Journal(WSJ) Garofolo et al. (1993a) and TIMIT Garofolo et al. (1993b) datasets, while pre-training was done on the subset of the LibriSpeech Panayotov et al. (2015) and WSJ datasets. The TIMIT dataset was used to mostly train phoneme recognition, while WSJ and LibriSpeech datasets were used for training the model in extensive speech tasks, like Word prediction. They also utilised the wav2letter++ toolkit for building and evaluating acoustic models, where some fine-tuning was done to prevent overfitting. The wav2vec model shows substantial improvements over the baseline systems it was compared to. Particularly on the WSJ nov92 test, the wav2vec large model trained on LibriSpeech got a WER of 2.43%, which outperforms most of the traditional models used at the time. The model also showed significant improvements in low-resource settings, having up to 36% reduction in WER. The authors also point out that using larger and more diverse datasets to train the wav2vec model leads to better outcomes. The results show that wav2vec is a good model for scenarios where labelled data is scarce. This makes the wav2vec model very useful in our situation, where we are trying to perform transfer learning for low-resource languages.

Conneau et al. (2020) also explores the use of unlabeled speech data across multiple languages to improve speech recognition performance, especially for low-resource languages. The authors use pre-existing architecture and training methods of wav2vec 2.0 and build upon it. They then make use of unsupervised learning to pre-train a model called XLSR from raw speech from multiple languages. As labelled data is scarce and it is even more the case in low-resource languages, this paper tries to solve that by using multiple languages to pre-trained an XLSR model.

XLSR made by authors adapts the wav2vec 2.0 framework to work with multiple languages, where it can learn from raw audio of different languages without the need for labelled data. The model architecture 2.2 is made of a convolutional feature encoder that maps the raw audio to latent speech representations, followed by a Transformer network outputting context representations. After that a quantisation module discretises these feature encoder outputs, forming the multilingual quantised latent speech representations. All of these allow the model to have a contrastive learning objective where it can learn to identify the correct latent representation among distractors. Following this, the model is pre-trained on unlabeled data from a mix of languages, aiming to capture universal speech features. Finally, for fine-tuning, a classifier is added on top of the model, which is then trained on labelled data using a Connectionist Temporal Classification (CTC) loss Graves and Graves (2012).

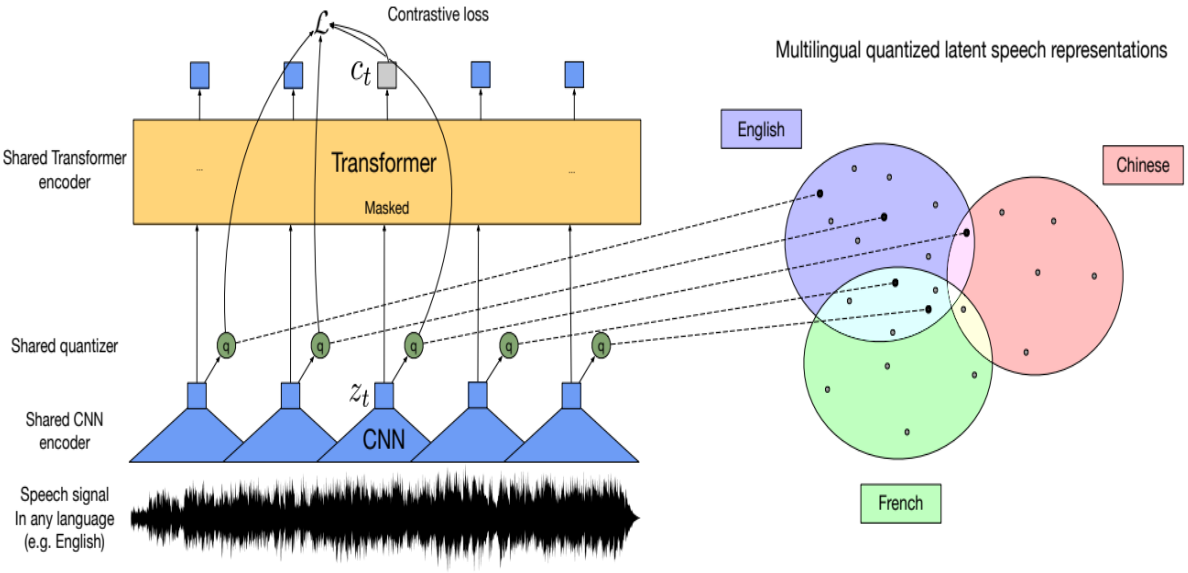


Figure 2.2: The XLSR model (Conneau et al. (2020))

The paper by Conneau et al. (2020) shows that cross-lingual pre-training significantly

outperforms monolingual approaches, particularly benefiting low-resource languages with a small amount of labelled data. The XLSR model achieves remarkable reductions in Phoneme Error Rate (PER) and WER across various benchmarks while using the Common Voice and BABEL datasets for training. This suggests that the model learns language-specific features and also captures general phonetic patterns that can be used across languages. This in our case means that the model effectively transfers knowledge between related languages and can be used for transfer learning in low-resource languages.

2.2.4 Improved Meta-Learning for Speech Recognition

Singh et al. (2022) introduces a framework that uses meta-learning to improve the speech recognition capabilities of ASR models for low-resource languages. End-to-End (E2E) models can often fall short when used for recognising low-resource languages due to the lack of labelled data. To counteract this, the authors propose that using the multi-step loss (MSL) function for the meta-learning framework could lead to more stable training procedures. Therefore, this would result in more accurate ASR systems for low-resource languages. The ASR model used in this paper is based on a Transformer architecture. It utilises a sequence-to-sequence model based on the encoder-decoder setup. The model also has a feature extraction layer that uses learnable VGG-based CNN when processing the input speech data. For the meta-learning component to quickly adapt to new languages with minimal gradient descent steps, the authors designed it to optimise the initialisation of model parameters. The Multi-Step Loss (MSL) method used by the authors complemented this by allowing a stable meta-learning process. To test the resulting model authors used the Common Voice v7.0 Ardila et al. (2019) dataset, which has multiple low-resource languages. The MSL-enhanced meta-learning framework significantly outperforms the standard MAML in terms of stability and accuracy. The results show improvements in Character Error Rates (CER) across multiple languages that were tested. We can see that MSL stabilises the training process of the ASR model and also accelerates convergence speeds and improves the accuracy of the model.

2.3 Summary

When working on developing ASRs for low-resource languages it is important to keep in mind the existing research in the same area. From the extensive research done in the field, we can gather the best methods and models researchers came up with and use them to propel our research. It is important to cite and give credit to the work done by other researchers, as it shows appreciation and allows that work to reach even more people.

We should also put a lot of consideration when designing our models and methods as to which pre-existing techniques to use based on their advantages and disadvantages.

I took a look at different ASR models, such as wav2vec Schneider et al. (2019) and XLSR Conneau et al. (2020) model. I also described the Transformer Dong et al. (2018) model, together with the Connectionist Temporal Classification(CTC) Lonergan et al. (2023). These models can be considered State of the Art models available for use. It is important to take into account the work done by the authors of these models when designing our own models and methodologies. All in all, in this section I described found methods and descriptions of work that have helped to design the proposed ASR system and helped to avoid the pitfalls researchers have faced in the past.

Chapter 3

Design

Before we can start the implementation, we have to plan and design our project based on the challenges and opportunities we identified during the literature review. We also need to determine what tools we will use for our implementation. So in this chapter, I will determine what kind of software needs to be built for our ASR system and discuss what key resources are needed.

3.1 Problem Formulation

The goal we have is to understand how viable transfer learning is for the low-resource language of Lithuanian. From this stems our main problem: how can we find the best pre-trained models to effectively learn low-resource languages without having to fine-tune models? It is computationally expensive to fine-tune large ASR models. Due to this, we are trying to find out if we can use some less computationally expensive methods, like clustering, to find the best pre-trained models. Then we would fine-tune those models to evaluate If our hypothesis is correct.

3.1.1 Identified Challenges

From what we saw in the previous chapter, one of the challenges we are facing is fine-tuning a model pre-trained on high-resource languages with data from low-resource languages. The main challenge would be the scarcity of labelled audio datasets in the Lithuanian language. Even with the use of transfer learning it might still be hard to fine-tune large models to fit the Lithuanian language. Another issue is the complexity of the Lithuanian language, where even native speakers often struggle with spelling and correctly building sentences. This is due to the Lithuanian language being mostly unchanged since the 14th century and also having quite a big alphabet, with multiple variations of similar

vowels. To counteract this we will need to perform some dataset pre-processing. The last challenge is the computational intensity of fine-tuning models as a lot of the training on the models is done on a GPU and requires quite a bit of memory. For this reason, I needed to devise ways to reduce the training workload and use techniques to speed up the training processes.

3.1.2 Proposed Work

Based on the identified challenges, I identified three main tasks:

- The First part would be collecting the Lithuanian language dataset, cleaning the dataset and preparing the dataset for future work with the models.
- The Second part would be feeding these datasets to various pre-trained models, extracting the features from the models and then performing clustering on them. Lastly, we will see if we can find similarities between the features of the pre-trained language and the target language without fine-tuning (reducing computational expense).
- The third part of this analysis is to fine-tune the pre-trained models and check if the results we got in the second part correlate with the fine-tuning performance.

3.2 Overview of the Design

3.2.1 Tools to be used

Jupyter notebook with Python kernel

The choice of programming language and the environment was the first thing that needed to be decided for the implementation of the project. For the environment of the project, I used Jupyter notebooks with a Python kernel. The main reason I chose the Jupyter notebook as the environment is because it provides an interactive environment where code, output, and notes are displayed in the same place, making it ideal for data analysis and iterative coding. Due to this, adjustments can be easily made on the fly after running a part of the code and receiving immediate feedback. Finally, I chose Python as my Programming language, as it is very versatile and has a massive ecosystem of different libraries and tools, such as Transformers, Torch, etc.

Libraries and Tools

Now that we will be using Python as our programming language, we need to explore the available libraries that will help us build our ASR system. The Transformers library Wolf et al. (2019) made by Hugging Face is perfect for this purpose, as it is easy to use and has state-of-the-art models for ASR. It includes models like Wav2Vec2, Wav2Vec2ForCTC, etc. that are pre-trained on vast datasets which is perfect in our case, as we need pre-trained models for transfer learning. Another Hugging Face library I will be using is Datasets. It is highly compatible with the Transformers library and is designed to easily load and pre-process data for machine learning. It has a great caching mechanism, which together with Jupyter notebook works seamlessly at speeding up data processing workflows. For part two 3.2.3 of my design, I used the Scikit-learn Pedregosa et al. (2011) library, as it has clustering and dimensionality reduction functionality. Another library for visualisation of the extracted features is Seaborn Waskom (2021), as it can handle complex datasets and create visualisations with relatively simple code. The last major library I used is Torch Collobert et al. (2011) as it has a strong GPU support, which greatly reduces training and fine-tuning time.

3.2.2 Part one: Dataset pre-processing

Firstly, I used the Mozilla-foundation Common Voice Lithuanian language audio dataset. After loading the dataset using the Dataset library from Hugging Face, I removed the columns not used in this analysis, special characters and extra vowel characters found in Lithuanian. Then I extracted the dictionary of unique characters present in the vocabulary. I did this to build a tokeniser that would be used in model training. I also set the sampling rate to 16000Hz, as most models only accept audio of this sampling rate. The last bit of the dataset manipulation was done to discard any utterance that was too large to load into the model given the GPU available to me.

3.2.3 Part two: Extracting model features and analysing them

To extract features from the pre-trained model, we first feed it the Lithuanian language dataset. Followed by taking the features from the last three layers of the model. Getting these facilitated analyses and visualisations of the features using different clustering techniques. I ran two clustering models, one being PCA with Agglomerative clustering and the second one being PCA with K-means clustering. I hypothesise that the higher the silhouette score of these clustering models and the more distinct the clusters are in visualisations, the better the results of fine-tuning the model with a low-resource Lithuanian

language dataset will be.

3.2.4 Part three: Fine-tuning pre-trained models

Fine-tuning the pre-trained models on the low-resource Lithuanian language dataset is the last step in the process. To fine-tune the models I initialised the training arguments and a Trainer model from the Transformers library Wolf et al. (2019) and then ran the Trainer using the pre-trained model.

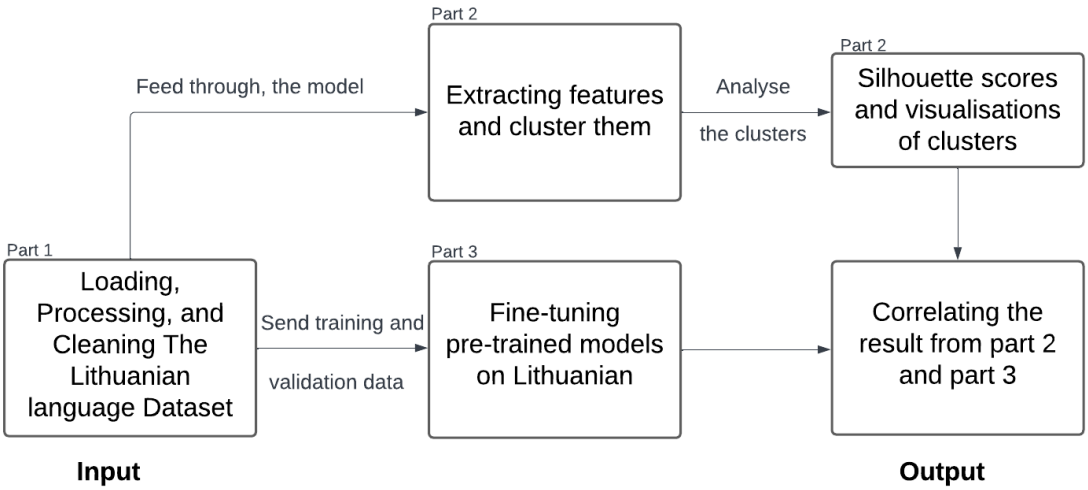


Figure 3.1: Diagram of Design

The final design is visualised in 3.1. It shows all the designed parts and the order in which they were run. The results of Part Two (3.2.3) and Part Three (3.2.4). will be compared in Section 5.2.

3.3 Summary

After considering and evaluating the best solutions, I provided a design for this project implementation. I also chose the most suitable programming language - Python, environment - Jupyter notebook and libraries, such as Transformers. Finally, I planned and explained each part of our design and created a visualisation showing the flow of the proposed design in the diagram 3.1.

Chapter 4

Implementation

4.1 Overview of the Solution

In the previous section, I proposed the design for the implementation and we outlined the steps needed to build our software system. All of the software implementation is done in a Jupyter notebook with a Python kernel in a sequential manner. The implementation follows the design 3.1, where the software has three parts. The first part is the loading of the Lithuanian Language dataset and cleaning it up into a format that models can accept. The second part is feeding these datasets into pre-trained Wav2Vec2 models that have yet to be fine-tuned and then extracting the features from the last three layers of the models. After extracting the features we need to perform clustering on it, followed by visualising those clusters and getting their silhouette scores. The third part consists of fine-tuning the Wav2Vec2 models, performing the predictions based on the test dataset and then calculating the Word Error Rate (WER) and Character Error Rate (CER). The last step is to evaluate the findings from parts two and three and see if there is any correlation.

4.2 Dataset Pre-procressing

I started the data pre-processing, by loading the Mozilla-foundation Common Voice Lithuanian language audio dataset Ardila et al. (2019). After loading the dataset I got rid of any metadata that were not used in the model, these include: accent, age, client_id, down_votes, gender, locale, segment and up_votes. The Lithuanian language dataset is too scarce to make any sense of the metadata so it was removed, as data points are often missing key metadata. Next, special characters were removed and the text was converted into lowercase. Even though large ASRs can predict special characters that constitute the punctuation in sentences, they need a language model (rules) to do so. Special characters

can negatively affect fine-tuning as it is much harder to classify speech segments into special characters as they don't correspond to a phoneme. Where the letter 'a' might have a clear sound, the special character "." does not 4.1.

```
chars_to_remove_regex = '"‘"„-[\,\?\.\!\-\;\:\"\\%\\\\\\']'

def remove_special_characters(batch):
    batch["sentence"] = re.sub(chars_to_remove_regex, '', batch["sentence"
    ]).lower()
    return batch

common_voice_train = common_voice_train.map(remove_special_characters)
common_voice_val = common_voice_val.map(remove_special_characters)
common_voice_test = common_voice_test.map(remove_special_characters)
```

Listing 4.1: Regex class to remove special characters and turn the sentences into lower case

The Lithuanian alphabet contains special characters, which can add extra complexity when fine-tuning models. These special characters are modifications of standard characters, some of which like 'Č', 'Š' and 'Ž' have different phonemes compared to C, S and Z. However this is not the case for the vowels, where modified vowels are just longer versions of standard characters. For example: 'ą' doesn't have a different sound from 'a', the sound is just longer, and one could say 'ą' == 'aa'. For this reason, I chose to replace these special characters with their standard counterparts 4.2, as there is no real phoneme difference between these characters. I replace these vowels to make it easier to fine-tune the models and increase the accuracy of the fine-tuned ASR.

```
def replace_extra_vowel_characters(batch):
    batch["sentence"] = re.sub('ą', 'a', batch["sentence"])
    batch["sentence"] = re.sub('į', 'i', batch["sentence"])
    batch["sentence"] = re.sub('y', 'i', batch["sentence"])
    batch["sentence"] = re.sub('ų', 'u', batch["sentence"])
    batch["sentence"] = re.sub('ū', 'u', batch["sentence"])
    batch["sentence"] = re.sub('ę', 'e', batch["sentence"])
    batch["sentence"] = re.sub('ė', 'e', batch["sentence"])
    return batch

```

```
common_voice_train = common_voice_train.map(replace_extra_vowel_characters)
common_voice_val = common_voice_val.map(replace_extra_vowel_characters)
common_voice_test = common_voice_test.map(replace_extra_vowel_characters)
```

Listing 4.2: Replacing special vowels in Lithuanian language

Next, I extracted unique characters from the dataset and saved them into a vocabulary dictionary. Then, I replaced the space " " with the "|" pipe character and also added '[UNK]' and '[PAD]' tokens into the vocab dictionary. I created this vocabulary dictionary, as it was used to initialise the CTC Tokenizer class from Wav2Vec2. This tokenizer was then used to fine-tune the model, changing the model output sizes to the length of the vocab dictionary.

```
def prepare_dataset(batch):
    audio = batch["audio"]

    batch["input_values"] = processor(audio["array"], sampling_rate=audio["
    sampling_rate"]).input_values[0]
    batch["input_length"] = len(batch["input_values"])

    with processor.as_target_processor():
        batch["labels"] = processor(batch["sentence"]).input_ids
    return batch

common_voice_train = common_voice_train.map(prepare_dataset, remove_columns
    =common_voice_train.column_names)
common_voice_val = common_voice_val.map(prepare_dataset, remove_columns=
    common_voice_val.column_names)
common_voice_test = common_voice_test.map(prepare_dataset, remove_columns=
    common_voice_test.column_names)
```

Listing 4.3: Processing the data into the format expected by Wav2Vec2ForCTC

Until now, I discussed the changes made to the metadata and labels of the dataset, now it's time to process the audio data. The first thing I changed is the sampling rate of the audio, as Common Voice audio data has a sampling rate of 48000Hz. The Wav2Vec2 models use audio data with a sampling rate of 16000Hz, so I re-sampled the audio data. Next, I used Wav2Vec2Processor to process the data to the format expected by Wav2Vec2ForCTC for

training 4.3. Finally, I filter out any audio with an input length of more than 90,000, to prevent Out-Of-Memory errors.

4.3 Extraction and analysis of model features

Now that the dataset has been cleaned, we can load in the pre-trained models and feed our dataset into it, to extract model features. For the models, I chose an English model by Grosman (2021a), the Facebook model by Arun et al. (2021) and the Russian model by Grosman (2021b). After choosing the models, I loaded them using the Wav2Vec2ForCTC method from the Transformers library 4.4 with parameters from the blog by von Platen (2021). Lastly, I set output_hidden_states to True in the model initialisation in order to view and extract features from model layers.

```
model = Wav2Vec2ForCTC.from_pretrained(
    "jonatasgrosman/wav2vec2-large-xlsr-53-russian",
    attention_dropout=0.1,
    hidden_dropout=0.1,
    feat_proj_dropout=0.1,
    mask_time_prob=0.05,
    layerdrop=0.05,
    ctc_loss_reduction="mean",
    pad_token_id=processor.tokenizer.pad_token_id,
    vocab_size=len(processor.tokenizer),
    output_hidden_states=True, # We add this to take a look at the features extracted!
    ignore_mismatched_sizes=True
)
```

Listing 4.4: Initialisation of the pre-trained models

Following this we feed input values into the pre-trained model using the Wav2Vec2Processor method from the Transformer library. Then we access and save the features from the hidden states of the last layer, second last layer and third layer. We then normalise the features we got using the MinMaxScaler method from the Scikit-learn library. We then perform Principal Component Analysis (PCA) with 4 components to reduce the dimensionality of the data. With the results from the PCA, I ran two different clustering methods: Agglomerative clustering and K-means clustering. Following this, I visualise the results by using the labels from the clustering methods and we also calculate silhouette

scores for each clustering.

4.4 Fine-tuning of pre-trained models

To start fine-tuning the models detailed above, we need to define a data collator von Platen (2021). This data collator treats input values and labels differently, applying different padding functions to them, as input and output are of different modalities. It also pads the training batches dynamically meaning that all training samples should only be padded to the longest sample in their batch, reducing computational expense and the amount of memory needed to run this fine-tuning. I also created a function that computes the WER metric and then uses it to evaluate the fine-tuning. Next, we need to freeze the feature extractor, so it doesn't get fine-tuned. We do this because the feature extractor consists of a stack of CNN layers that extract "acoustically meaningful - but contextually independent - features" von Platen (2021) from the raw audio data.

We proceed to define all the training arguments 4.5, noting that a small batch size was used due to memory constraints. I also batch inputs of similar size together, dodging a lot of useless padding and speeding up the training process. Additionally, I assigned the gradient accumulation parameter to a higher value as we have smaller batch sizes. This allows for the simulation of larger batch sizes without the corresponding increase in memory usage, where instead of updating model weights after each batch, the accumulated gradient of multiple smaller batches is averaged and used as model weights. Then we initialise a trainer that takes in all the instances of classes and functions we have made 4.6. The model is then fine-tuned in the Lithuanian language. The test dataset was then passed into this model and the audio input was transcribed into a text output. Lastly, we take the transcribed audio returned by the model and the labels from the test dataset to calculate the WER and CER metrics.

```
training_args = TrainingArguments(
  output_dir=r".",
  group_by_length=True,
  per_device_train_batch_size=2,
  per_device_eval_batch_size=1,
  gradient_accumulation_steps=4,
  evaluation_strategy="steps",
  num_train_epochs=10,
```

```
    gradient_checkpointing=True,
    save_steps=200,
    eval_steps=200,
    logging_steps=1000,
    learning_rate=3e-4,
    warmup_steps=10,
    save_total_limit=2,
)
```

Listing 4.5: Initialisation of Training arguments arguments

```
trainer = Trainer(
    model=model,
    data_collator=data_collator,
    args=training_args,
    compute_metrics=compute_metrics,
    train_dataset=common_voice_train,
    eval_dataset=common_voice_val,
    tokenizer=processor.feature_extractor,
)
```

Listing 4.6: Initialisation of the Trainer

4.5 Summary

In this chapter, I presented a detailed description of the software system I built based on the design I proposed in the previous Section 3. I also explained how I manipulated the dataset for optimal results and why I made the changes I did. Then I proceeded to explain how the features were extracted from the model by feeding it the input dictionary. I outlined the reasons why I needed to do this and what results I wanted to achieve by clustering these features. Lastly, I explained the fine-tuning process, the reasons why I chose certain parameters for training and what results we are trying to calculate from the fine-tuned model. Having completed the implementation of the software we can proceed to the Evaluation stage, where I will discuss the experiments and the final results.

Chapter 5

Evaluation

Evaluation in this case consists of two separate experiments. The first experiment will involve clustering the extracted features, getting the silhouette scores and analyzing the visualisations of clusters. For the second experiment, I present the recognition results for the test dataset and calculate the Word Error Rate (WER) and Character Error Rate (CER) metrics. Lastly, I show whether there is any correlation between the results of the two experiments, which will indicate whether we can use clustering to find languages that have similar features. If true, this could reduce the computational expense necessary for developing ASR models and make Transfer-learning more accessible for low-resource languages.

5.1 Experiments

For these experiments, I chose three different models from Hugging Face: an English model Grosman (2021a), a multilingual Facebook model Arun et al. (2021) and a Russian model Grosman (2021b). The English model is a large XLSR model pre-trained in 53 languages and fine-tuned for speech recognition in English. The Facebook model is a large XLSR model trained in 128 different languages and has 300m parameters. Lastly, the Russian model similar to the English model is a large XLSR model pre-trained on 53 languages and fine-tuned for speech recognition in Russian. For training, validation and testing datasets on the Lithuanian language, I used the Common Voice Corpus 17.0 Ardila et al. (2019). This dataset has 24 validated hours of transcribed speech recordings in the Lithuanian language. We are going to use this dataset for both of our experiments, for the first experiment I am going to use only the test dataset, while for the second experiment, I will use all three datasets: training, validation and test.

5.1.1 Experiment One: Silhouette scores and Visualisation of feature clustering

For the first experiment, I decided to test the last three layers of the models we loaded from pre-trained. The experiment started by loading in the pre-trained models using the Wav2Cev2ForCTC method from Transformers library **??**. Next, I fed the model the pre-processed test dataset, to get the model outputs. I then extracted the features from the last three layers of the model, in order to cluster them. Before clustering, I normalised the activations using the MinMaxScaler method from Scikit-learn. Normalising the features obtained from the models was important because it brought all the features to the same scale, allowing the clustering methods to converge faster with better performance. Another reason to do this, was because we are using the K-means algorithm to perform clustering, which computes the distance between data points. If the features are not scaled, the K-means algorithm can perform poorly on this task.

After the features have been retrieved and normalised, we perform clustering on them. The first thing we do is reduce the dimensionality by using Principal Component Analysis (PCA) and taking the best four components. We then cluster the data from PCA, compute the silhouette scores and visualise the clusters using the Seaborn library. As part of the first experiment, I obtained silhouette scores and visualised the clusters. To measure how long it took to run the clustering algorithms and obtain silhouette scores I used Jupyter notebook's in-built system which measures the time to run a piece of code's execution within a cell. It is important to note that I ran the first experiment on my PC, which has an AMD Ryzen 5900X 12-core CPU, with 64GB of DDR4 RAM so the speeds may differ based on the machine being used.

5.1.2 Experiment Two: WER and CER for fine-tuned models

For the second experiment, I calculated the WER and CER metrics after the test dataset was passed into a fine-tuned model. For this experiment, we will use the same models as in the first experiment. We will use the train and validation split of the Common Voice dataset to train and evaluate the training process of the model **??**. After fine-tuning, I passed the test dataset to get speech recognition results (predictions) and then calculated WER and CER metrics by testing those results against the labels. I also measured the time needed to train each model on the Lithuanian language, by using Jupyter notebook's in-built system which measures the time to run a cell with code. To train the models I used my laptop with a Nvidia RTX 3070 with 8 GB of onboard memory and 8 more GB of shared memory.

5.2 Results

In this section, I will be talking about the results obtained in the two experiments, their significance and how the results from experiment 1 and experiment 2 correlate.

5.2.1 Results of experiment one

We can see the silhouette scores for the last three layers with both clustering methods and for all three models in our Table 5.1. The first thing we see in the results is that PCA and K-means get better silhouette scores compared to PCA and Agglomerative clustering. We can also notice how the silhouette score of each layer differs from one another and based on the model either the third or the last layer performs the best. This way the English model has a silhouette score of 0.489 for the third layer and a silhouette score of 0.454 for the last layer. Similarly, the Russian model also has the best results on the third layer equalling 0.430, meanwhile, the multilingual Facebook model has the best results on the last layer - 0.384. From these results, we can see that the Facebook model performs the worst, which is surprising considering it has the largest number of parameters and has been trained in 128 different languages. The English and Russian model scores are not far from each other, but the English model silhouette scores are slightly higher. Now we can take a look at the runtime of the clustering for each model in Table 5.2. The main takeaway from the runtime is that PCA with K-means is much faster than PCA with agglomerative clustering. Considering that agglomerative clustering also produces worse results compared to K-means, it does not make sense to keep using agglomerative clustering for evaluating the clusters of these model features.

Table 5.1: Silhouette Scores

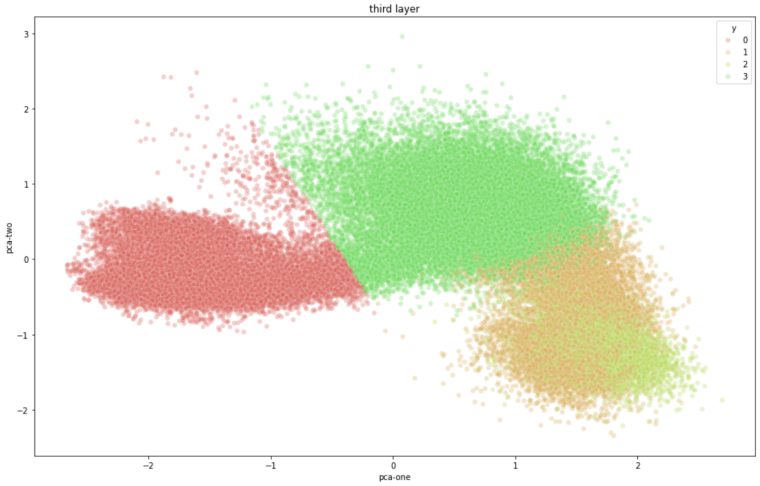
Methods	English Model	Facebook Model	Russian Model
Last layer PCA + Kmeans	0.45391539	0.38363495	0.42095956
Second last layer PCA + Kmeans	0.45993826	0.35967001	0.38149124
Third layer PCA + Kmeans	0.48879614	0.36589018	0.43029657
Last layer Agglomerative + PCA	0.18018568	0.12158866	0.12297121
Second last layer Agglomerative + PCA	0.16219062	0.14129034	0.09873130
Third layer Agglomerative + PCA	0.15105152	0.11973694	0.10230178

Table 5.2: Runtime of clustering and silhouette score calculations

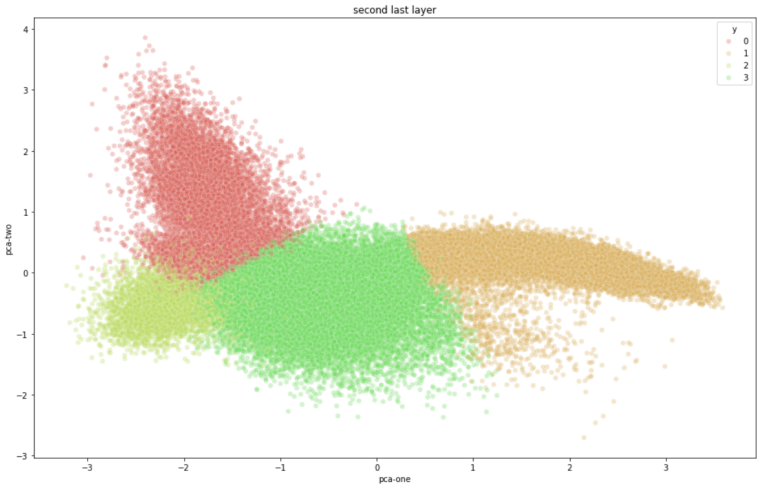
Methods	English Model	Facebook Model	Russian Model
Last layer PCA + Kmeans	1m 6s	1m 19s	1m 9s
Second last layer PCA + Kmeans	1m 12s	1m 22s	1m 17s
Third layer PCA + Kmeans	1m 3s	1m 15s	1m 7s
Last layer Agglomerative + PCA	22m 43s	25m 12s	23m 7s
Second last layer Agglomerative + PCA	23m 52s	27m 37s	20m 56s
Third layer Agglomerative + PCA	21m 47s	25m 11s	22m 36s

As agglomerative clustering doesn't perform the best, I only visualise the clusters created from PCA + K-means. First, I will discuss the visualisation of the clusters in Figure 5.1 for the English model. In Figure 5.1(a) of the third layer, we can see some distinct clusters but there is a large degree of overlap between clusters. Considering the silhouette scores obtained for this layer I would expect better-defined clusters. However, in the second last and last layer visualisations we can see some distinct clusters where we can distinguish between the clusters with the naked eye. This could mean that the network can distinguish between the languages it is decoding. In the last layer scatter plot, we can also see a cluster disconnected from other clusters, indicating some sort of structure found in the features.

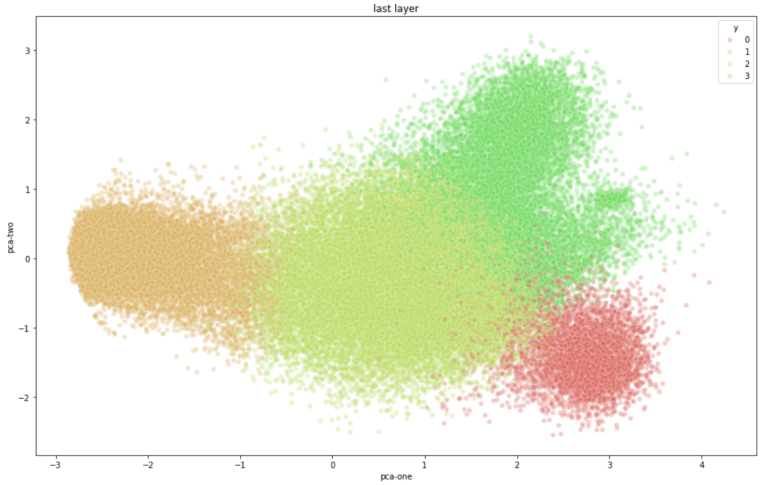
In Figure 5.2, which corresponds to the features of the multilingual Facebook model, we can see a clear separation between the clusters. Even though there is a clear separation between clusters, it's hard to tell if the clusters have distinct features, as they have a continuous transition from one cluster to another. Lastly, in the visualisation of the clusters 5.3 of the Russian model, we can see some distinct groupings of data. Some of the clusters seem to have a higher density of data points, which could be indicative of the way the underlying data is distributed.



(a) Third Layer

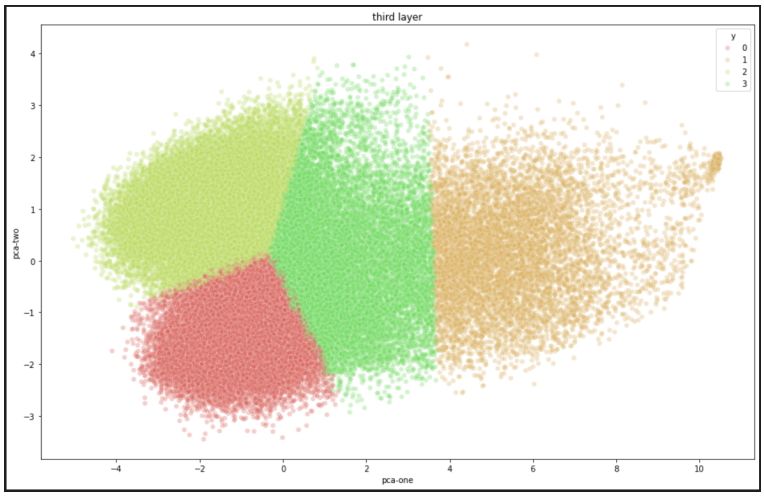


(b) Second Last Layer

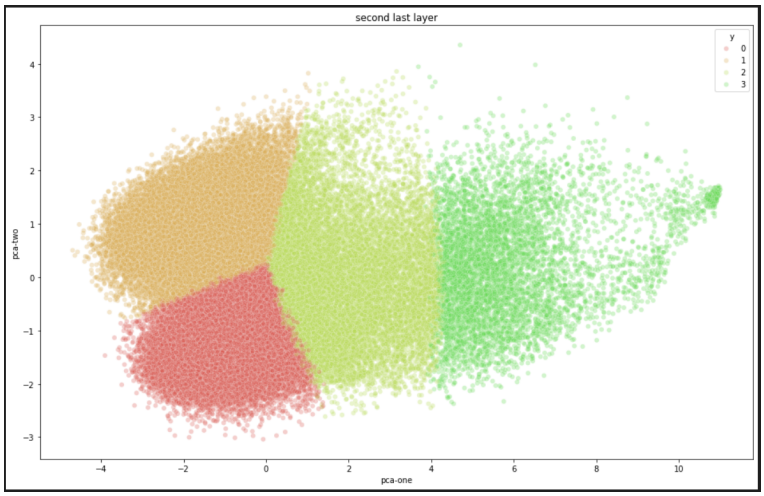


(c) Last Layer

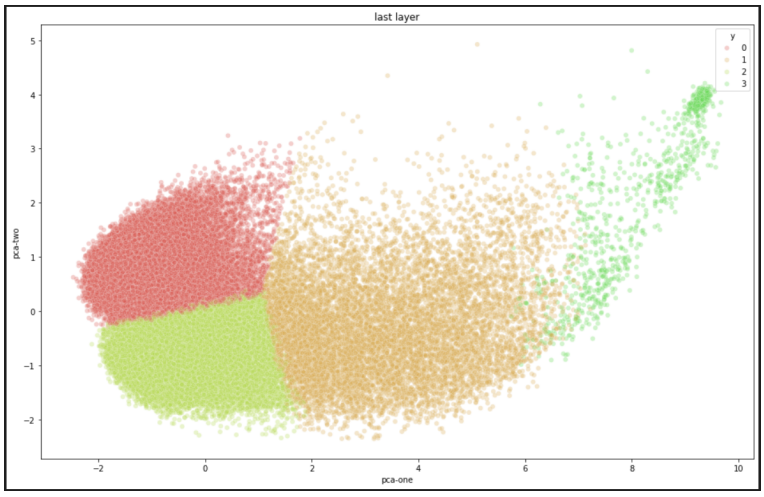
Figure 5.1: Visualisations of clusters for the English model using PCA + Kmeans



(a) Third Layer

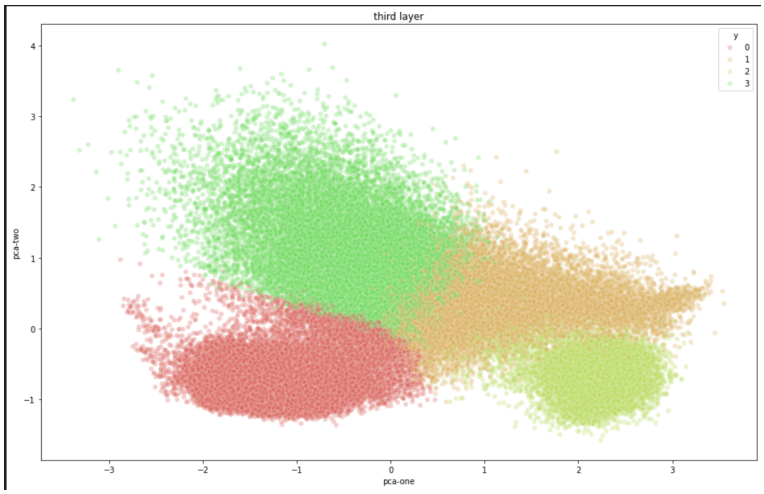


(b) Second Last Layer

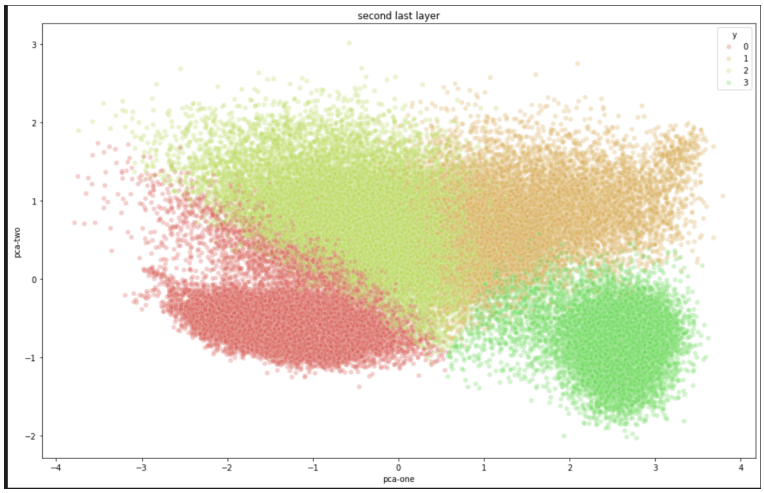


(c) Last Layer

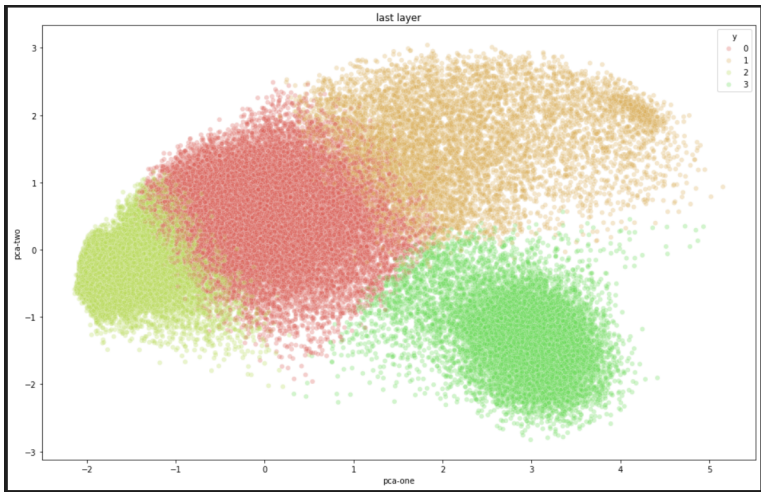
Figure 5.2: Visualisations of clusters for the Facebook model using PCA + Kmeans



(a) Third Layer



(b) Second Last Layer



(c) Last Layer

Figure 5.3: Visualisations of clusters for the Russian model using PCA + Kmeans

5.2.2 Results of experiment two

In the Table 5.3 we can see the Word Error Rates (WERs) and Character Error Rates (CERs) obtained after fine-tuning the models and getting the predictions. From the results, we can see that the Russian model was the most effective in speech recognition after it went through fine-tuning. English closely followed, with a score of 0.73% in terms of WER and 0.19% in CER. The multilingual Facebook model ended up getting the worst results, which is surprising as it is the largest model out of the three and was trained in 128 languages. Based on the research from Conneau et al. (2020), we would assume that the more languages a model has been pre-trained on, the better performance it would have after it is fine-tuned for a low-resource language. We should also notice that the models have low CER, showing that fine-tuned models can recognise characters with a surprising degree of accuracy. Meanwhile, the WER rate is quite high, but considering it's fine-tuned from the pre-trained model of another language, this is unsurprising. We should also consider that the Lithuanian dataset it was trained on is quite scarce and the training was done for 10 epochs - around 220 minutes. Overall the results obtained from fine-tuning were reasonably good, especially the CER being around 14%.

Table 5.3: WER and CER scores for our fine-tuned models

Model	**WER**	**CER**	**Training time**
English xlsr-53	62.19	14.56	231m
Facebook xls-r-300m	67.17	16.66	219m
Russian xlsr-53	61.46	14.37	223m

5.3 Summary

For this section, I described two experiments and discussed the results obtained from them. I also discussed what these results entail. At this point, we should also discuss the association between the results from the first and second experiments. The results of the first experiment suggested that the multilingual Facebook model would have the worst performance based on its low silhouette scores. From the results of the second experiment, we can see that this indeed ended up being true. However, the silhouette scores also showed that the English model would end up with the best performance. However, the English model lead to slightly worse WER and CER results than the Russian model, which ended up having the best performance. This could be explained by the visualisations of the clusters from experiment one, where the Russian model had the most distinct groupings of data and had a higher density of data at those distinct feature clusters.

Chapter 6

Conclusions & Future Work

I started this work by researching the literature in the area of automatic speech recognition, transfer learning and learning for low-resource languages. I also researched works done in the field, such as XLSR Conneau et al. (2020) and Wav2Vec Schneider et al. (2019), which introduce unsupervised learning for speech recognition. I talked about how this work can be utilised to build the proposed ASR system while avoiding the pitfalls researchers faced in the past. I then identified the problems with developing such a model, the ways they were solved and the tools used. Following these, I proposed the design of a software system, which is divided into three parts: Dataset pre-processing, model feature extraction and analysis, and lastly, fine-tuning of pre-trained models. For the next step, I implemented our software system from the proposed design. Firstly, I started our work in a Jupyter notebook with Python Kernel, we loaded the Common Voice Lithuanian language dataset Ardila et al. (2019) and cleaned up the training, validation and test datasets. Secondly, I initialised the models 4.4, created methods to extract the features and performed clustering on the features. Then I obtained the silhouette scores and visualised the clusters obtained from the extracted features. Lastly for the implementation, I initialised the trainer arguments 4.5 and the trainer 4.6 itself and then fine-tuned the pre-trained models that were loaded. Following this I calculated the Word Error Rate (WER) and Character Error Rate (CER) from the predictions we got from the fine-tuned models.

With the software system ready, I performed two major experiments. I started by loading the pre-trained models and feeding the Lithuanian language dataset into the models. I followed this by extracting the features from the last three layers of the model and performing clustering on normalised feature values. As the last part of experiment one, I got the silhouette scores and visualised the clusters from these clustering methods.

Next, I performed experiment number two, where we fine-tuned the pre-trained models on Lithuanian and then got the WER and CER values. Given the CER results, I concluded that transfer learning is a viable and efficient approach for enhancing ASR systems in low-resource languages. I also observed some association between the results of the first and second, where the fine-tuned large Facebook model got the worst WER and CER, just as the silhouette scores indicated. Although the silhouette scores of the English model were slightly higher than the Russian model, they ended up having very similar WER and CER scores. This can be explained by the more favourable cluster visualisations of the Russian model where it has the most distinct groupings of data and higher density of data at those distinct feature clusters. From these findings, I conclude that we can reduce the computational expenses of choosing models by pinpointing model compatibility prior to fine-tuning.

6.1 Future Work

There are three main directions for future work: apply this software system to more models, test it in different low-resource languages and perform similar analyses for other ASR network architectures. By exploring these in future works we could further enhance transfer learning by applying it to a broader number of languages. This would open up the path for low-resource languages to have better speech recognition systems and would allow for larger linguistic diversity across the international scene.

Bibliography

Ardila, R., Branson, M., Davis, K., Henretty, M., Kohler, M., Meyer, J., Morais, R., Saunders, L., Tyers, F. M., and Weber, G. (2019). Common voice: A massively-multilingual speech corpus. *arXiv preprint arXiv:1912.06670*.

Arun, B., Wang, C., Tjandra, A., Lakhotia, K., Xu, Q., Goyal, N., Singh, K., von Platen, P., Saraf, Y., Pino, J., Baevski, A., Conneau, A., and Auli, M. (2021). Facebook's wav2vec2 xls-r counting 300 million parameters. `https://huggingface.co/facebook/wav2vec2-xls-r-300m`.

Bansal, S., Kamper, H., Livescu, K., Lopez, A., and Goldwater, S. (2018). Pre-training on high-resource speech recognition improves low-resource speech-to-text translation. *arXiv preprint arXiv:1809.01431*.

Bender, E. M., Gebru, T., McMillan-Major, A., and Shmitchell, S. (2021). On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*, pages 610–623.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

Black, A. W. (2019). Cmu wilderness multilingual speech dataset. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5971–5975. IEEE.

Collobert, R., Kavukcuoglu, K., and Farabet, C. (2011). Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*.

Conneau, A., Baevski, A., Collobert, R., Mohamed, A., and Auli, M. (2020). Unsupervised cross-lingual representation learning for speech recognition. *arXiv preprint arXiv:2006.13979*.

Davis, H. (1952). The listening audrey.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dong, L., Xu, S., and Xu, B. (2018). Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5884–5888. IEEE.

Garofolo, J., Graff, D., Paul, D., and Pallett, D. (1993a). Csr-i (wsj0) complete ldc93s6a. *Web Download. Philadelphia: Linguistic Data Consortium*, 83.

Garofolo, J., Lamel, L., Fisher, W., Fiscus, J., Pallett, D., Dahlgren, N., and Zue, V. (1993b). Timit acoustic-phonetic continuous speech corpus ldc93s1. web download. philadelphia: Linguistic data consortium.[electronic database].

Godard, P., Adda, G., Adda-Decker, M., Benjumea, J., Besacier, L., Cooper-Leavitt, J., Kouarata, G.-N., Lamel, L., Maynard, H., Müller, M., et al. (2017). A very low resource language speech corpus for computational language documentation experiments. *arXiv preprint arXiv:1710.03501*.

Godfrey, J. and Holliman, E. (1993). Switchboard-1 release 2 ldc97s62. *Linguistic Data Consortium*, 34.

Graff, D., Huang, S., Cartagena, I., Walker, K., and Cieri, C. (2010). Fisher spanish speech (ldc2010s01). *Web Download. Philadelphia: Linguistic Data Consortium*.

Graves, A. and Graves, A. (2012). Connectionist temporal classification. *Supervised sequence labelling with recurrent neural networks*, pages 61–93.

Grosman, J. (2021a). Fine-tuned XLSR-53 large model for speech recognition in English. `https://huggingface.co/jonatasgrosman/wav2vec2-large-xlsr-53-english`.

Grosman, J. (2021b). Fine-tuned XLSR-53 large model for speech recognition in Russian. `https://huggingface.co/jonatasgrosman/wav2vec2-large-xlsr-53-russian`.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., et al. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507.

Li, X., Metze, F., Mortensen, D. R., Black, A. W., and Watanabe, S. (2022a). Asr2k: Speech recognition for around 2000 languages without audio. *arXiv preprint arXiv:2209.02842*.

Li, X., Metze, F., Mortensen, D. R., Watanabe, S., and Black, A. W. (2022b). Zero-shot learning for grapheme to phoneme conversion with language ensemble. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2106–2115.

Lonergan, L., Qian, M., Chiaráin, N. N., Gobl, C., and Chasaide, A. N. (2023). Towards dialect-inclusive recognition in a low-resource language: are balanced corpora the answer? *arXiv preprint arXiv:2307.07295*.

Mortensen, D. R., Li, X., Littell, P., Michaud, A., Rijhwani, S., Anastasopoulos, A., Black, A. W., Metze, F., and Neubig, G. (2020). Allovera: A multilingual allophone database. *arXiv preprint arXiv:2004.08031*.

Panayotov, V., Chen, G., Povey, D., and Khudanpur, S. (2015). Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5206–5210. IEEE.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830.

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Schneider, S., Baevski, A., Collobert, R., and Auli, M. (2019). wav2vec: Unsupervised pre-training for speech recognition. *arXiv preprint arXiv:1904.05862*.

Schultz, T. (2002). Globalphone: a multilingual speech and text database developed at karlsruhe university. In *Interspeech*, volume 2, pages 345–348.

Singh, S., Wang, R., and Hou, F. (2022). Improved meta learning for low resource speech recognition. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4798–4802. IEEE.

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

von Platen, P. (2021). Fine-tuning xls-r for multi-lingual asr with transformers. `https://huggingface.co/blog/fine-tune-xlsr-wav2vec2`.

Waskom, M. L. (2021). seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. (2019). Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.