Speech Representations

Adventures in pre-training and fine-tuning transformers for speech technology tasks

Nik Vaessen

Institute for Computing and Information Sciences

RADBOUD UNIVERSITY PRESS

Radboud Dissertation Series

Speech Representations

Adventures in pre-training and fine-tuning transformers for speech technology tasks

Nik Vaessen

Speech Representations
Adventures in pre-training and fine-tuning transformers for speech technology tasks
Nik Vaessen

Radboud Dissertation Series
ISSN: 2950-2772 (Online); 2950-2780 (Print)

Published by RADBOUD UNIVERSITY PRESS
Postbus 9100, 6500 HA Nijmegen, The Netherlands
www.radbouduniversitypress.nl

Design: Nik Vaessen
Cover: Nik Vaessen and Proefschrift AIO | Guntra
Printing: DPN Rikken/Pumbo

ISBN: 9789465151090
DOI: 10.54195/9789465151090
Free download at: https://doi.org/10.54195/789465151090

RADBOUD
UNIVERSITY
PRESS

Speech Representations

Adventures in pre-training and fine-tuning transformers for speech technology tasks

Proefschrift

ter verkrijging van de graad van doctor
aan de Radboud Universiteit Nijmegen
op gezag van de rector magnificus prof. dr. J.M. Sanders,
volgens besluit van het college voor promoties
in het openbaar te verdedigen op

woensdag 10 september 2025
om 10:30 uur precies

door

Nik Vaessen

geboren op 9 april 1996
te Heerlen

Promotor

Prof. dr. ir. D.A. van Leeuwen

Copromoter

Dr. T.M. van Laarhoven

Manuscriptcommissie

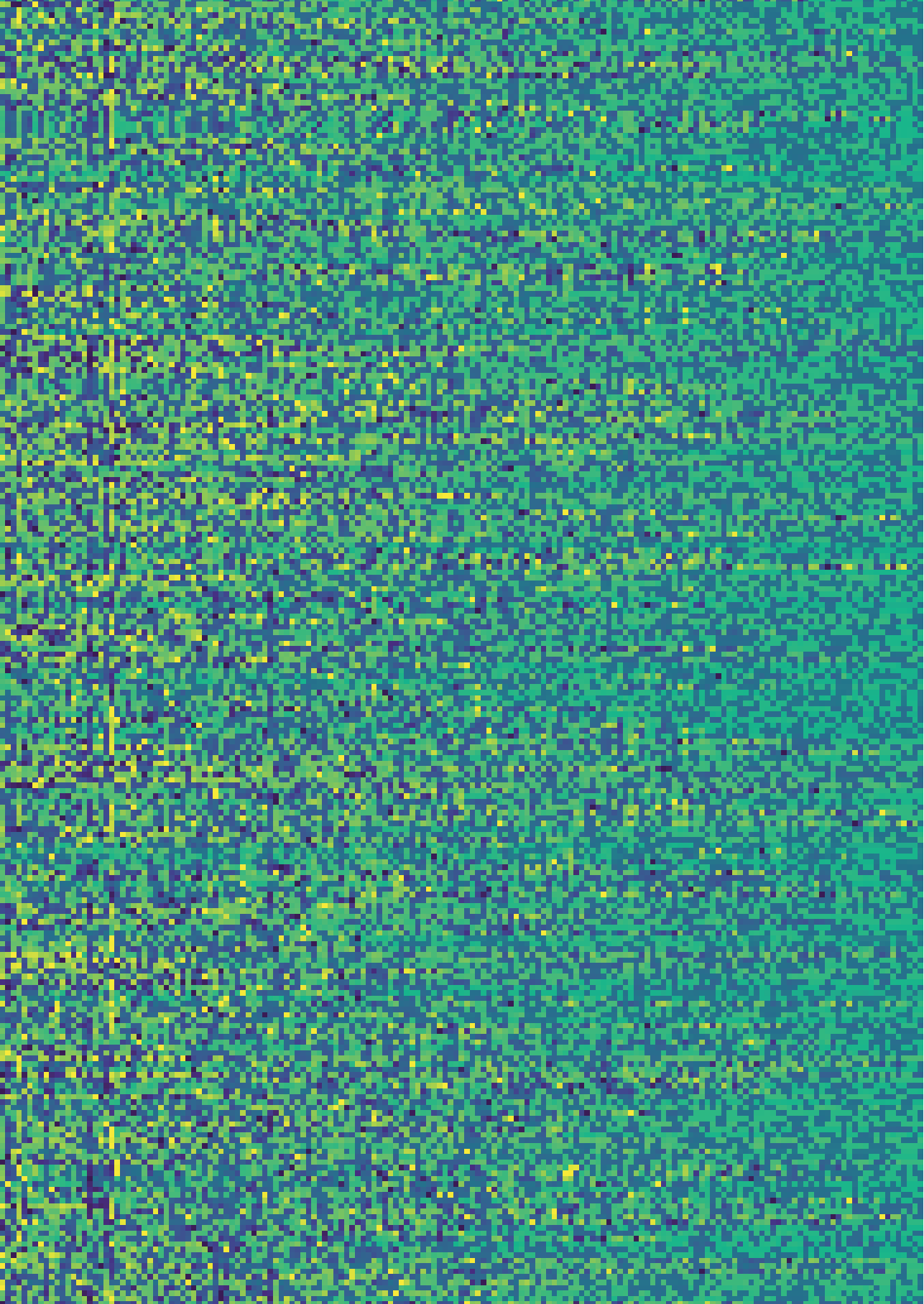
Prof. dr. M.A. Larson
Prof. dr. ir. H. Van hamme (KU Leuven, België)
Dr. J.A. Rohdin (Vysoké učení technické v Brně, Tsjechië)

Ter ere van

Tonny Kusters

En u, *mijn moeder*, door mij zo geacht,
Vloek, zegen mij met tranen, maar vecht door.
Verdwijn niet zomaar in de zoete nacht.
Vecht, vecht, omdat het licht niet sterven mag.

Table of contents



Samenvatting

Stel je voor dat je iemand moet uitleggen hoe je de stem van je moeder herkent aan de telefoon, of hoe je kunt verstaan wat een vriend zegt in een druk café. Omdat deze dingen vanzelf gaan, kunnen we niet echt uitleggen hoe we dat doen. Dit betekent dat we ook niet weten hoe we een computer moeten instrueren om sprekers te herkennen of spraak uit te schrijven. In plaats daarvan vertrouwen we op *machinaal leren*, waarbij we een computer de instructies geven om patronen in data te onthouden, en deze patronen later toe te passen op nieuwe, onvoorziene situaties.

Dit proefschrift vertelt een avontuur over zelfgestuurd leren van spraakrepresentaties – een vorm van machinaal leren, en een mooie manier om te zeggen "computers leren spraak te begrijpen door ze zelf dingen uit te laten zoeken." Een vergelijking kan gemaakt worden met hoe kinderen taal leren: niemand legt elk woord dat ze horen uit; in plaats daarvan pikken ze patronen op natuurlijke wijze op in het dagelijks leven.

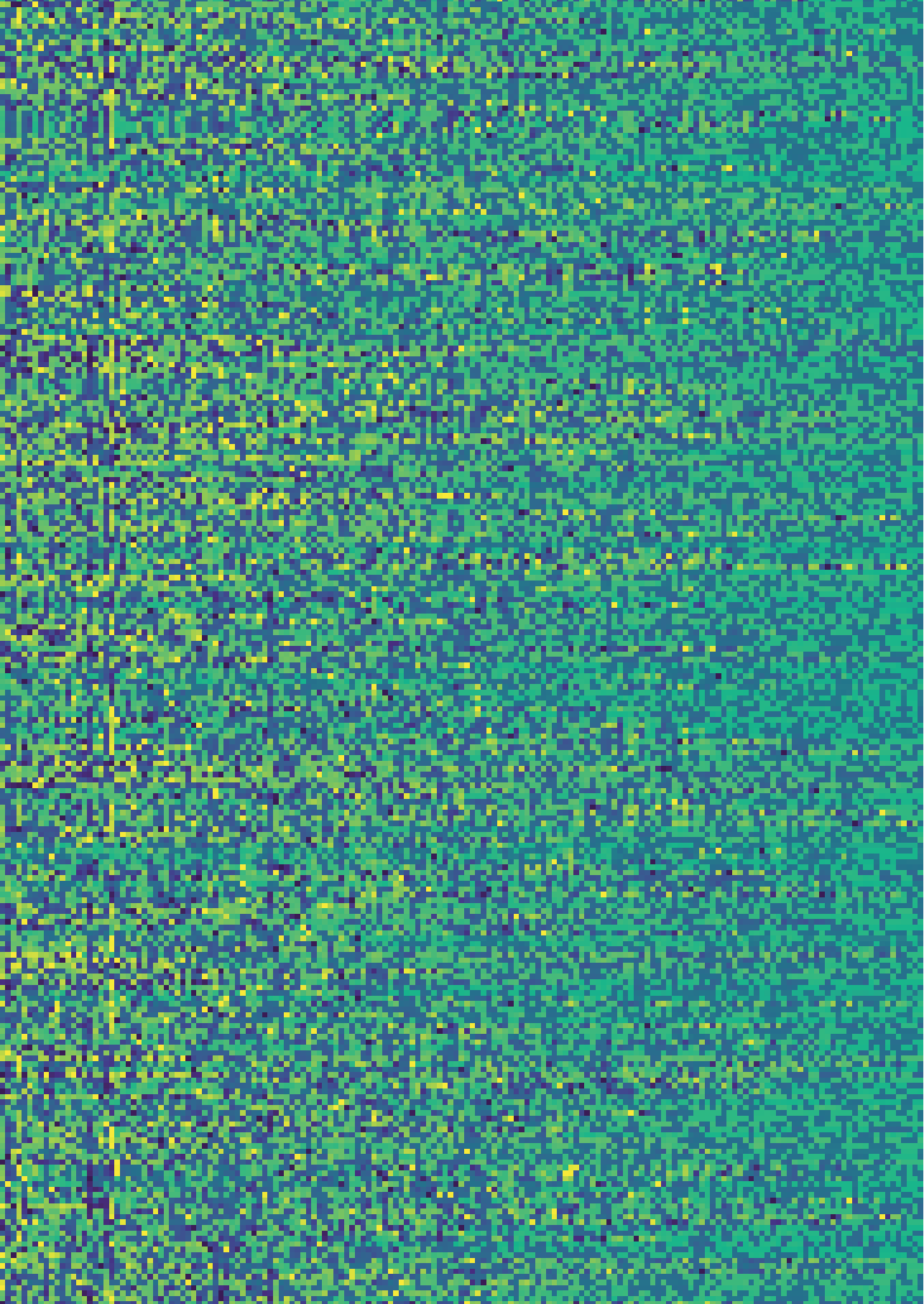
Het avontuur begint met een neurale netwerkarchitectuur genaamd *wav2vec 2.0*. Dit computerprogramma gebruikt het concept van zelfsturing om basisfeiten over spraak te leren. Deze basisfeiten dienen als springplank, waardoor het netwerk aangepast kan worden voor taken zoals het herkennen van Engelse spraak, met minimale menselijke begeleiding. Het blijkt dat deze basisfeiten het ook mogelijk maken om wav2vec 2.0 aan te passen voor andere spraaktechnologietaken. Onze eerste studie bevestigde dat aanpassing voor *sprekerherkenning* mogelijk was. In deze studie gebruikten we als begeleiding echter wel veel voorbeelden van spraak die gekoppeld waren aan een specifiek individu. Een tweede studie onthulde dat het aanpassen ook kon worden gedaan met minder menselijke begeleiding, waarbij nog steeds betere capaciteiten werden waargenomen, ten opzichte van andere neurale netwerkarchitecturen zonder zelfsturing.

Niet alles werkte echter zoals gehoopt. Toen we probeerden wav2vec 2.0 aan te passen zodat het *tegelijkertijd* kon herkennen wie er spreekt én wat er gezegd wordt, ontdekten we dat de twee taken elkaar hinderen. Dit is niet helemaal verrassend, aangezien spraakherkenning moet werken ongeacht wie iets zegt, terwijl sprekerherkenning moet werken ongeacht wat er gezegd wordt. Het concept van zelfgestuurd leren omzeilt dit fundamentele spraaktechnologieprobleem vooralsnog niet.

Ondanks deze successen blijft zelfsturing computationeel duur. De auteurs van wav2vec 2.0 gebruikten clusters van gespecialiseerde computers die uren aan spraak tegelijk kunnen verwerken. Is dit strikt noodzakelijk? Heb je dure supercomputers nodig om een netwerk zoals wav2vec 2.0 te bouwen? We ontdekten dat dit *niet* het geval is. Dezelfde zelfsturingsmethode kan gebruikt worden met slechts één gewone

computer – het type met een grafische kaart dat veel mensen hebben voor gaming of videobewerking. Het nadeel is dat het langer duurt: in plaats van het netwerk in een dag of twee te kunnen bouwen met veel computers, duurt het tot wel een maand op één computer.

Ten slotte eindigt het avontuur met het bouwen van een nieuwe versie van wav2vec 2.0 specifiek voor de Nederlandse taal. Samen met het Nederlands Instituut voor Beeld en Geluid verzamelden we een enorme hoeveelheid Nederlandse spraak uit televisie-uitzendingen, verspreid over 50 jaar (1972-2022). We gebruikten deze data voor de zelfsturing van het nieuwe wav2vec 2.0 netwerk. Bij het vergelijken van ons Nederlands-georiënteerde netwerk met andere versies, gebouwd met tientallen talen, ontdekten we dat specialisatie in één taal zijn voordelen heeft. Het netwerk dat alleen met Nederlandse spraak was gebouwd, bleek robuuster wanneer het onbekende Nederlandse spraak tegenkwam.



Summary

Imagine trying to explain to someone how you recognize your mother's voice on the phone, or how you can understand what a friend is saying even in a noisy café. As these things come naturally, we cannot really explain how to do so. This means we also do not know how to instruct a computer to recognize speakers or transcribe speech. Instead, we rely on *machine learning*, where we give a computer the instructions to remember patterns in data, and then later apply these patterns to new, unforeseen scenarios.

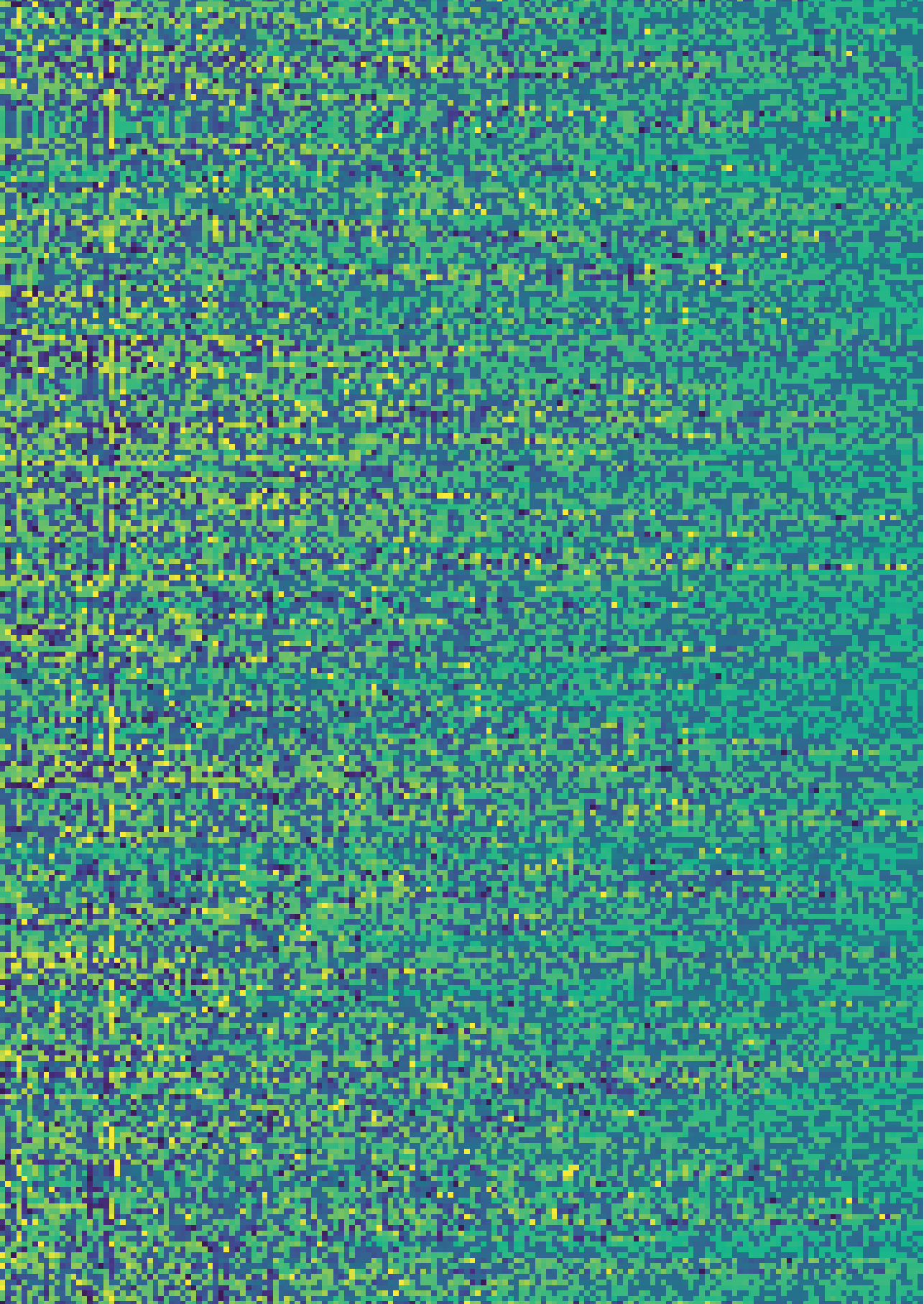
This thesis narrates an adventure into self-supervised speech representation learning – a form of machine learning, and a fancy way of saying "teaching computers to understand speech by letting them figure things out on their own." A comparison can be made to how children learn language: they do not need someone to explain every word they hear; instead, they pick up patterns naturally through their daily life.

The adventure begins with a neural network architecture called *wav2vec 2.0*. This computer program uses the concept of self-supervision to learn basic facts about speech. These basic facts serve as a stepping stone, allowing the network to be adapted for new tasks, like recognizing English speech, with minimal human guidance. It turns out that these basic facts also allow wav2vec 2.0 to be adapted to other speech technology tasks. Our first study confirmed that adaptation for *speaker recognition* was possible. However, this study used a lot examples of speech tied to specific individuals. A second study revealed that this adaptation was also feasible with less human guidance, with better capabilities compared to neural network architectures not using self-supervision.

Not everything worked as hoped, though. When we tried to adapt wav2vec 2.0 so that it could *simultaneously* recognize both who's speaking and what they're saying, we found that the two tasks hinder each other. This is not completely surprising, as speech recognition should work well regardless of who said something, while speaker recognition should work regardless of what was said. The concept of self-supervised learning does not circumvent this fundamental speech technology problem yet.

Despite these successes, self-supervision remains computationally expensive. The authors of wav2vec 2.0 used clusters of specialized computers that can process hours of speech at once. Is this strictly required? Do you need expensive supercomputers to build a network like wav2vec 2.0? We found that this is *not* the case. The same self-supervision method can be used with just one regular computer – the type with a graphics card that many people have for gaming or video editing. The catch is that it takes longer: instead of training the system for a day or two on many computers, you might need to train it for a month on one computer.

Finally, the adventure ends with building a new version of wav2vec 2.0 specifically for the Dutch language. Together with the Netherlands Institute for Sound and Vision, we collected a massive amount of Dutch speech from television broadcasts, spanning 50 years (1972-2022). We used this to self-supervise wav2vec 2.0. Comparing our Dutch-focused model against models trained on dozens of languages, we found that specializing in a single language has its advantages. The Dutch-only model proved more robust when encountering unfamiliar Dutch speech.



1 Introduction

In which our adventurer acquaints thee with their quest to uncover knowledge about speech representations.

Artificial neural networks have taken the world by storm. In 2024, two Nobel prizes were awarded to research on this specific technology; John J. Hopfield and Geoffrey Hinton won the Nobel Physics prize "*for foundational discoveries and inventions that enable machine learning with artificial neural networks*" (The Royal Swedish Academy of Sciences, 2024a), while David Baker, Demis Hassabis, and John Jumper won the Nobel Chemistry prize "*for protein structure prediction*" (The Royal Swedish Academy of Sciences, 2024b) with their work on AlphaFold (Jumper et al., 2021), a family of artificial neural networks which can be used to analyze protein folding behavior. Besides research, consumers have also started to explicitly use software products of which the main component is a (very large) artificial neural network. In September 2024, the Dutch Central Bureau of Statistics (CBS) reported that 23 % of the Dutch population has used tools like ChatGPT (Ouyang et al., 2022). In the age group of 18 to 25 years this percentage even increases to 49 %. Implicitly, deep learning technology has become ubiquitous, as many human-computer interactions involve software with artificial neural network components, including recommendation engines (e.g., TikTok, YouTube), search engines (e.g., Google), translation tools, medical diagnosis, and last but not least, interfacing with computational devices using speech (e.g., Apple's Siri and Amazon's Alexa).

However, speech interfaces are only one use-case of artificial neural networks in the field of speech technology. In this dissertation, we present research applying artificial neural networks to speech, building towards *a singular neural network* with broad usage capabilities in various problems involving speech, similar to how large language models like ChatGPT generalize to many text-based tasks. In this introduction, we will first give a brief overview of these speech technology problems. Further, we will highlight one of the leading paradigms in deep learning, *representation learning*, with a framework of (self-supervised) pre-training and fine-tuning artificial neural networks. Then, we will explain how this framework differs between the domains of text, images, and speech, and what challenges arise within the speech context. Finally, we can give an overview of the contributions of this manuscript to the application of artificial neural networks in speech technology.

1.1 The breadth of speech technology

In this section, we distinguish speech technology tasks into 5 categories, broadly following the classification in SUPERB (Speech processing Universal PERformance Benchmark) (Yang et al., 2021). The first and obvious category involves the *content* of the speech, or "what" is said. Here, *automatic speech recognition* (ASR) is the task which most people are familiar with due to, e.g., the speech interface software mentioned above. ASR involves the mapping of the speech signal to linguistic units, often phonemes, letters, or words. The ASR task can be seen as overarching, with subcategories that practitioners can consider in isolation. One subcategory is based on different sensor conditions, e.g., far-field microphones, stereo microphones, or an audio-visual signal. Another subcategory consists of particularly challenging domains, e.g., air traffic control, speech of children, or pathological speech. Lastly, language can be seen as a subcategory, as almost all research is conducted on English audio. Another content-based task is *keyword spotting*. This task involves the recognition of a particular phrase, often used to wake-up a device, but it can also be carried out in forensic or military context. In the wake-up setting, keyword spotting is often heavily constrained by the computation budget, as it should be executed continuously, on a device with a power-efficient (thus, slower) CPU, without a GPU, and without sending the audio signal to a remote server.

The second category involves the *speaker* of the speech signal, in other words, "who" said it. The *speaker recognition* task is about determining the identity of the person that spoke. Generally, this task involves the comparison of an enrollment segment (known speaker) and a test segment (speaker has to be determined). This trial of enrollment and test segments is scored based on two likelihoods: whether the utterances come from the same speaker or from different speakers. The calibration of a threshold is required to decide whether a particular score implies the test segment is the same or a different speaker. Another task is *speaker diarization*. Here, the absolute identity of the speaker is not of interest. Instead, a model needs to recognize how many speakers are in a speech signal, and determine, at any given time step, which of these detected speakers are speaking, if any.

The third category encompasses the *prosody* of speech, i.e., "how" it was said. This task can be described as *emotion recognition*, with a focus on classifying the emotional state of the speaker. Collecting data for this task is challenging, as emotions are difficult to capture, and for data-collection purposes they are often acted out instead of naturally expressed. Moreover, nuances such as sarcasm, or a particular intonation which, e.g., expresses doubt, are difficult to express in text. Thus, one can also consider enhancing ASR output with the detection and labeling of this kind of prosodical information in speech.

The fourth category is on the *semantic* meaning of the speech, that is, "what is meant"? Here, *intent classification* involves directly mapping a speech signal to an intended outcome. Alternatively, it is possible to use an ASR system to produce

text, and then use a text-based natural language understanding model. However, in this case, errors from the ASR system can propagate, which can negatively affect the performance compared to a direct approach. The *slot filling* task is related, which should determine the information about the intended action, e.g., an alarm should be set *for 6 o'clock in the morning*. The *speech translation* task involves transcribing and translating a speech signal at the same time, which requires semantic understanding. Similar to intent classification, this can also be accomplished with a two-step approach, but there are benefits to direct translation.

The last category is on the generation or modification of speech. The *text-to-speech* task involves, given some text, and optionally, some speaker attributes, the generation of a speech signal containing said text with the given speaker attributes. Another task is *speech separation*. An example for this task is that a speech signal contains multiple speakers. This signal should be split into multiple signals, where each signal only has one speaker. Related, *speech enhancement* involves removing noise from a speech signal, such that only the actual speech remains.

There is a final task, which is important for any practitioner in the speech technology field, but which is difficult to place into one of the aforementioned categories. This is the task of *speech activity detection*, in other words, detecting whether a given audio signal actually contains speech. Although this task seems trivial for humans, it is difficult to build a system which is reliable in all possible circumstances. One reason for this is that it can be unclear how to exactly define speech. For example, it can be ambiguous whether singing, rap, or background speech should be seen as speech. Applying speech activity detection is often carried out as a pre-processing step so that only audio with speech is processed by a model.

1.2 The paradigm of pre-training and fine-tuning

The tasks in the previous section used to be solved with different signal processing and classical machine learning techniques, while the necessity of domain knowledge required a researcher to focus on one particular task. However, the advent of artificial neural networks has led to a singular method which can be applied to any task, usually improving performance at the same time. This enables research collaboration and mutual improvements. One of the important developments which enabled this wide application of artificial neural networks is the *pre-training and fine-tuning* paradigm. Artificial neural networks differentiate from classical machine learning methods by intrinsically learning *representations* instead of using hand-crafted features. Usually, a neural network trained on one particular task, has learned representations which are *also* useful for solving a different, related task. This paradigm was initially successful in computer vision, where neural networks (Krizhevsky et al., 2012; Simonyan and Zisserman, 2015; He et al., 2016) were first, thus "pre-", trained on the ImageNet dataset (Deng et al., 2009) to classify pictures. Then, it was observed that these trained image classification neural networks could be fine-tuned on another task, e.g.,

object detection on the COCO dataset (Lin et al., 2014). Fine-tuning consisted of simply initializing a neural network with the obtained weights from the ImageNet training, and then starting a new training, on another task, with a different dataset. One trick to make this fine-tune training work was to freeze, i.e., not update, certain parts of the network, usually the first few layers. This pre-training and fine-tuning paradigm has two benefits, namely, fine-tuning is much cheaper computationally, and at the same time, better performance is obtained than training on the other dataset from scratch.

Later, *self-supervised* pre-training was introduced. The idea behind self-supervision is based on one of the properties of artificial neural networks training, namely, compared to classical approaches, it is computationally feasible to drastically scale the dataset size, and usually, more data is better. However, increasing the amount of data is expensive if it requires human labeling. Self-supervised techniques use a *pretext* task, with automatic labels generated from the data samples. Examples of such automatic labels are:

1. transforming a color image into a grey-scale image, and then reconstructing the color image from the grey image.
2. removing a word form a sentence, and then predicting the word which was removed.
3. removing a segment of audio from a speech utterance, and then being able to select which segment was removed, when given the choice between a few options.

Self-supervision is beneficial because, in order to solve the pretext task, the neural network needs to represent the input data in such a way as to be able to perform the task. If the pretext is appropriately challenging, these representations generalize to "real" tasks. This is made apparent by self-supervised networks obtaining excellent performance after being fine-tuned, compared to random initializations.

1.3 What differentiates speech from text and images?

As stated above, (self-supervised) representation learning was initially made popular in the computer vision (CV) domain. For the natural language processing (NLP) domain, its use led to the development of the BERT model (Devlin et al., 2019) and the GPT models (Radford et al., 2018; 2019; Brown et al., 2020). It has become apparent that the speech community lags behind in the development of overarching systems, which can solve most tasks in the domain, like ChatGPT. The self-supervised methods, and the lessons learned therefrom, which led to the GPT model family, are not directly applicable to speech (Mohamed et al., 2022). These difficulties arise, when compared to CV or NLP:

1. Speech is a variable-length sequence. In CV, pretext tasks operate on a fixed-size image. In comparison, speech pretext tasks need to be flexible to the length of the signal.
2. Speech is a *long* sequence. In NLP, text is relatively short, e.g., this sentence is 82 characters long. When spoken, the previous sentence is around 7 seconds, implying

more than 100 k audio samples when recorded with the standard sampling rate of 16 kHz. This stark difference in length requires different modeling techniques.

3. Speech is difficult to segment into stand-alone units. In NLP, concrete units of meaning are easily separable, i.e., spacing and punctuation exist. Currently, for speech, there is no method to accurately segment an utterance into concrete units of meaning, be it phonemes, words, or sentences.
4. Speech does not have a clean vocabulary. In NLP, a common pre-text task is to predict the next word. This is relatively straightforward, as all words in the training dataset are known beforehand. There is no equivalent task (i.e., a set of discrete options to predict during training) in speech. It can be argued that phonemes are the vocabulary of speech, but predicting the next phoneme is a lot more difficult. It cannot be discretized, as the variability of a phoneme in the speech signal is quite large, due to, e.g., sensor noise and co-articulation.
5. Speech tasks are orthogonal, with different informational needs. For example, representations for ASR need to contain phonetic information, while speaker recognition requires information on speaker attributes. For NLP, a pretext task does not need to learn information relevant to the writing style of authors, as this is not required for tasks that are deemed relevant, e.g., question answering or named entity recognition. For speech, modeling speaker information is important, as, e.g., ASR benefits from speaker diarization capabilities, and text-to-speech models want to generate natural sounding voices.
6. Speech, like an image in CV, needs to be captured, thus, the raw signal is inherently noisy, and models need to be robust to varying sensors. In NLP, the closest equivalence is words which are new, or misspelled, but these can simply be filtered out. It is very unlikely that two recordings of a particular speaker saying a particular word lead to an identical speech signal.

Taken together, it is not surprising that methods for NLP and CV are not directly applicable for speech, and therefore progress is slower in comparison.

1.4 Thesis contributions

In this dissertation, we present research with the intent to gain insights about self-supervised speech representation learning, which, on a high level, can help bridge the gap between speech on one hand, and NLP and CV on the other hand. Each chapter is based on a stand-alone research article, which can lead to a slight difference in notation and style. This also means that each chapter will, to varying extent, repeat some details on, e.g., the wav2vec 2.0 architecture (Baevski et al., 2020a), and the Librispeech (Panayotov et al., 2015) and VoxCeleb2 (Chung et al., 2018) datasets. The contributions of each chapter are as follows.

In **Chapter 2** we present work on fine-tuning wav2vec 2.0, a self-supervised speech representation method, for the speaker recognition task. The original publication (Baevski et al., 2020a) only considered whether the learned speech representations

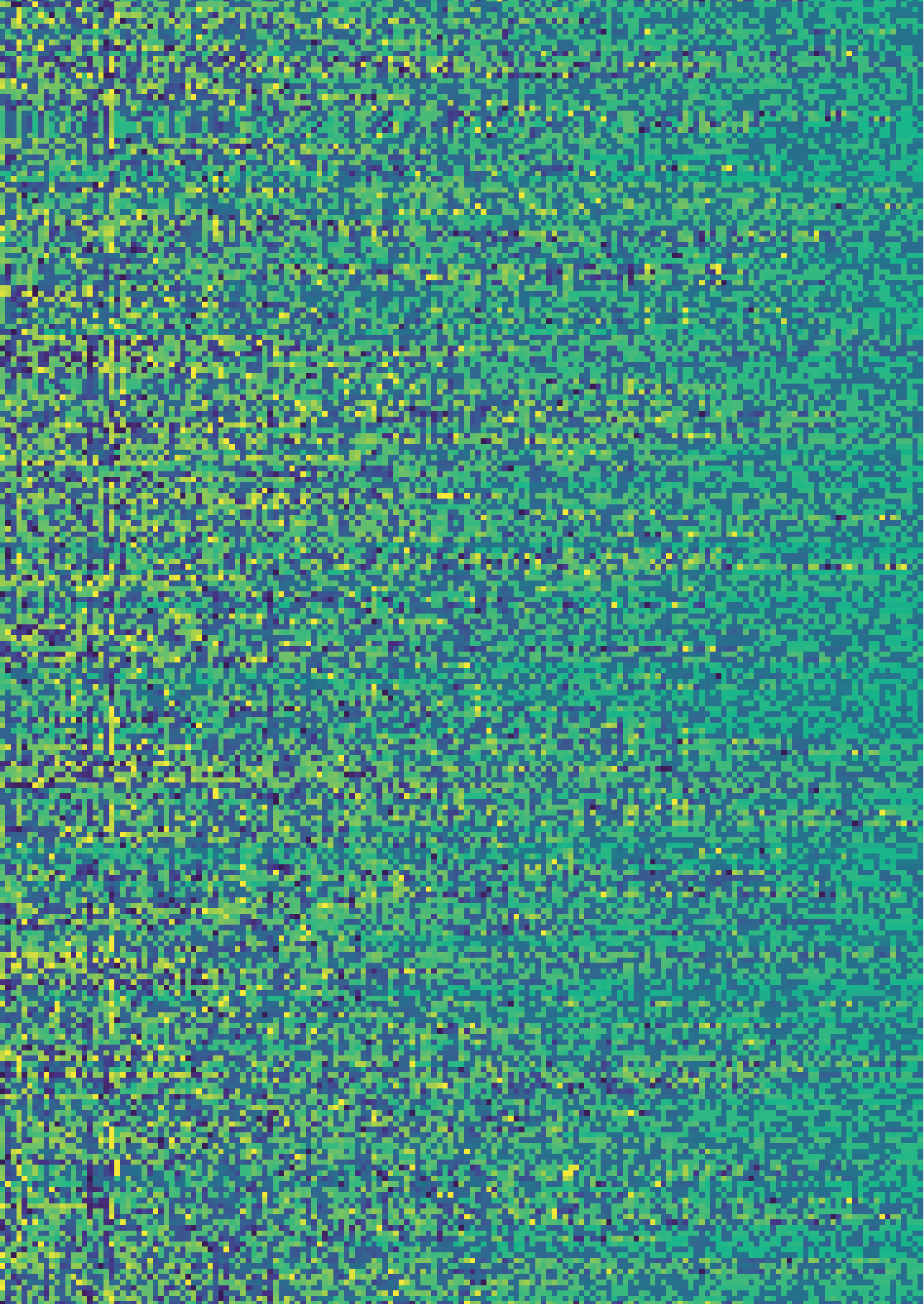
were applicable for automatic speech recognition. From the NLP domain, it was known that BERT (Devlin et al., 2019), which shared commonalities with wav2vec 2.0, could be fine-tuned for a wide variety of text-based tasks. This raised the question: How adaptable is wav2vec 2.0 to other speech tasks? We made evident that the speech representations from the pre-trained wav2vec 2.0 network can be adequately fine-tuned for speaker recognition.

One of the main findings in (Baevski et al., 2020a) about the proposed wav2vec 2.0 framework was that fine-tuning (for speech recognition) was feasible with a very limited amount of labeled data, in this case 10 minutes of transcribed audio. In **Chapter 3** we, similarly, study the performance when fine-tuning with a limited amount of labeled data, but *for speaker recognition* instead of ASR. Initial positive results were found when designing a project for master degree students following the *Machine Learning in Practice* course taught at Radboud University. We wanted to make a machine learning competition similar to those hosted on Kaggle, where students competed by building a model with the best speaker recognition performance on the VoxCeleb1 test set. We limited the size of the training dataset to make it more feasible to converge within the 10 hours of GPU time students were allotted to use per training run. When testing various baselines, so we knew how to guide students throughout the project, we found that fine-tuning wav2vec 2.0 gave the best performance in this setting, prompting further analysis. We made evident that the low-resource capabilities of wav2vec 2.0 are shared between different speech tasks. Furthermore we gained insights on which properties are beneficial to good fine-tuning performance in the (low-resource) training dataset.

Spurred by finding speaker recognition capabilities in wav2vec 2.0, we were interested in learning speech representations which had intrinsic phonetic and speaker information. When fine-tuning for speech recognition, the final network cannot be used for speaker recognition, and vice versa. In **Chapter 4**, we conduct experiments with *multi-task learning*, where we jointly fine-tune for speaker *and* speech recognition. The resulting model should then be able to provide a shared speaker and phonetic representation, which should ease development of speaker-attributed speech recognition, or speaker diarization, with only a single neural network. However, we learned that the speech and speaker recognition tasks are difficult to combine. First, there is no good dataset which has labels for both tasks. We show that using the speaker labels of Librispeech is insufficient to generalize to a well-performing speaker recognition network. To work around this dataset issue, we optimize with disjoint batches, where each optimization step consists of two forward steps, one with a speaker-labeled batch from a speaker recognition dataset, and the other one with a transcription-labeled batch from a speech recognition dataset. We found that disjoint training was not mutually beneficial. Although we saw mild improvements in speaker recognition performance, this was to the detriment of speech recognition performance. Moreover, we found that this disjoint training setup is not robust to out-of-domain data, with large degradations in performance compared to single-task learning.

Previous chapters have considered fine-tuning wav2vec 2.0. In the last two chapters, we will study aspects of self-supervised pre-training. In **Chapter 5**, we analyze the relationship between the batch size during pre-training, and downstream task performance. Self-supervised pre-training is very computationally expensive. While seminal work reports performances on their models trained with large batch sizes (which require many GPUs), they do not share results with low(er) computational budgets. This leaves open questions regarding small batch sizes, e.g., is there a lower bound on the batch size for self-supervision to converge? Moreover, what is the expected performance when a practitioner has a certain computational budget? In our analysis, we show that pre-training can be accomplished with small batch sizes, even those which can fit on a single consumer-grade GPU. We also show a direct relationship between the number of training epochs, and downstream task performance.

Lastly, in **Chapter 6** we compare pre-training with a mono-lingual and multi-lingual dataset. We create a large (55 k hours) Dutch dataset based on television broadcasts from 1972 to 2022, collected from the archives of the Netherlands Institute of Sound and Vision. To assist with the creation of this dataset, we also studied data quality assumptions for self-supervised pre-training. State-of-the-art speech recognition for Dutch is realized with multi-lingual models, such as Whisper (Radford et al., 2023) and wav2vec 2.0 XLSR (Conneau et al., 2021). We were interested in whether pre-training wav2vec 2.0 with a mono-lingual dataset, with roughly the same size as the multi-lingual dataset for XLSR, has better performance for Dutch. Moreover, seminal works in self-supervised speech representation learning do not make explicit which properties are needed in pre-training datasets. Knowledge of these properties can help practitioners who want to create a high-quality pre-training dataset. We found that clean data is important to stable wav2vec 2.0 pre-training. While small quantities of noise are acceptable, the presence of music is detrimental. Segmenting the raw broadcast dataset with Whisper was an effective strategy to pre-process data. We found that fine-tuning mono-lingual models leads to higher robustness on out-of-domain data.



2 Fine-tuning wav2vec 2.0 for speaker recognition

In which our adventurer uncovers the usefulness of speech representations, obtained with contrastive self-supervised learning, for the speaker recognition task.[1]

In the field of natural language processing (NLP) it has become standard to fine-tune self-supervised pre-trained models, such as BERT (Devlin et al., 2019), XLNet (Yang et al., 2019), and T5 (Raffel et al., 2020), on a wide variety of NLP tasks. Recently, this framework of pre-training and fine-tuning has also been successfully used in automatic speech recognition with wav2vec 2.0 (Baevski et al., 2020a).

The BERT and wav2vec 2.0 network have commonalities in their design. Firstly, they are both encoder-only transformer networks (Vaswani et al., 2017) with the exact same architectural setup for both the `BASE` (12 layers, hidden dimension of 768) and `LARGE` (24 layers, hidden dimension of 1024) variants. Secondly, they use self-supervised pre-training with masked input to learn representations, which can later be fine-tuned for downstream tasks. However, they differ in four major aspects:

1. The input tokens to the transformer layers in wav2vec 2.0 are speech features, from raw audio processed by a learned convolutional neural network, instead of WordPiece embeddings (Wu et al., 2016).
2. The self-supervised loss function for wav2vec 2.0 is contrastive, in that the model needs to distinguish, at a masked time step t, the true representation from a set of distractors. For BERT, the loss function is predictive, as the model can immediately predict the masked word with a classification over the entire vocabulary, which is known in advance.
3. Wav2vec 2.0 uses relative positional embeddings computed by a single convolutional layer instead of sinusoidal positional embeddings.
4. There is no "class" and "separator" token, nor a next-sentence prediction task, in the pre-training procedure of wav2vec 2.0.

The lack of these class and separator tokens raises questions about the adaptability of wav2vec 2.0 to downstream tasks other than speech recognition. The BERT model

[1]This chapter is based on the publication Vaessen, N., and van Leeuwen, D. A. (2022). Fine-Tuning Wav2Vec2 for Speaker Recognition., in *International Conference on Acoustics, Speech and Signal Processing*, 7967–7971. doi: 10.1109/ICASSP43922.2022.9746952.

is quite flexible, with four fine-tuning strategies, depending on the downstream task, which involve these class and separation tokens:

(i) Single-sentence classification tasks, e.g., sentiment analysis, give as input to the network, in order, the class token, and the sequence of WordPiece tokens of a sentence. The output sequence of the network is mostly ignored, only the class token is used as input to a linear layer for, e.g., binary classification.

(ii) Sentence-pair classification tasks, e.g., entailment, give as input to the network, in order, the class token, the sequence of WordPiece tokens of sentence A, the separation token, and the sequence of WordPiece tokens of sentence B. Similar to (i), a final linear layer uses the output class token for, e.g., a binary classification.

(iii) For the question-answering task, the input is, in order, the class token, the tokens of a question, the separator token, and the tokens of a paragraph which could contain an answer. Each output token of the paragraph is used to determine a start and end token spanning the answer to the question.

(iv) For a single-sentence tagging task, e.g., named entity recognition, the input is, in order, the class token, and all tokens of a sentence. The class token in the output sequence is not used. For all other tokens, a linear layer is used to classify each word(piece) independently.

Unlike BERT, the wav2vec 2.0 framework only supports a single output configuration equivalent to (iv). We speculate the absence of a class and separation token due to the fact that wav2vec 2.0 was originally designed to be fine-tuned (in low-resource settings) for speech recognition. A reasonable analogy can be made between sentence tagging and speech recognition, where each output token of wav2vec 2.0 can represent a phone or letter. It is unclear whether other speech technology tasks can also be modelled this way. We think that our task of interest, speaker recognition, should be seen as either a single-utterance classification task, similar to (i), or a utterance-pair classification task, similar to (ii). For our purposes, it could be required to have access to a class token which acts as a representation which summarizes the whole input sequence.

In this work we focus on the feasibility of fine-tuning wav2vec 2.0 for speaker recognition. We want to know whether wav2vec 2.0, pre-trained on Librispeech (Panayotov et al., 2015), is an effective initialization not only for speech recognition, but also for another speech technology task, namely speaker recognition. Concretely, we set out to answer the following research questions:

RQ 1. Can we fine-tune wav2vec 2.0 for speaker recognition by replacing BERT's concept of a class token with a pooling layer, thus modeling speaker recognition as single-sentence classification?

RQ 2. If so, what pooling method is the most effective?

RQ 3. Alternatively, can we fine-tune wav2vec 2.0 for speaker recognition by modelling it as a binary classification of a sentence pair?

The remainder of this chapter is set out as follows. We will first discuss related work in Section 2.1. Then, in Section 2.2 we detail the wav2vec 2.0 architecture and propose our adaptions for speaker recognition. The experimental setup is explained in Section 2.3 and results are shown in Section 2.4. Finally, we will discuss and conclude the experimental results in Section 2.5.

2.1 Related work

Initial evidence suggests that wav2vec 2.0 network can be applied to a variety of speech-related tasks. Concurrently to our work, Fan et al. (2021) use the network for speaker recognition and language identification in both a single and multi-task learning setting. Tjandra et al. (2022) show good performance for language identification with 25 languages, but modify wav2vec 2.0 to use log-mel spectrogram input instead of raw waveforms. In Pepino et al. (2021) the (frozen) wav2vec 2.0 embeddings are input to a learnable downstream model for emotion recognition. Meanwhile Yuan et al. (2021a) manages to fine-tune the wav2vec 2.0 model itself on emotion recognition with CTC loss by using emotion-labeled phonetic units. Finally, the LeBenchmark proposed by Evain et al. (2021) uses the wav2vec 2.0 models as a baseline and encapsulates speech recognition, spoken language understanding, emotion recognition and speech translation in a single benchmark.

2.2 Methodology

This section will describe the wav2vec 2.0 network architecture, the pre-training and fine-tuning procedure for the original speech recognition task, and will detail the required adaptations for the speaker recognition task.

2.2.1 The wav2vec 2.0 architecture

The wav2vec 2.0 framework (Baevski et al., 2020a) applies the concept of self-supervised pre-training with transformers to automatic speech recognition. In Figure 1 we show a general overview of the network architecture during fine-tuning. The next subsections summarize each component.

Feature extraction

The first step is to encode a raw audio waveform (normalized to zero mean and unit variance) into learned representations with a discrete time unit. The *feature extractor* consists of 7 consecutive 1-dimensional convolutions with 512 channels and respective kernel sizes of (10, 3, 3, 3, 3, 2, 2) and stride (5, 2, 2, 2, 2, 2, 2). The output of the first convolutional layer is group normalized (Wu and He, 2018) such that each of the 512 channel sequences has zero mean and unit variance before GELU activation (Hendrycks and Gimpel, 2016) is applied. The other convolutional layers do not have any normalization layers and their output is directly activated with GELU. The

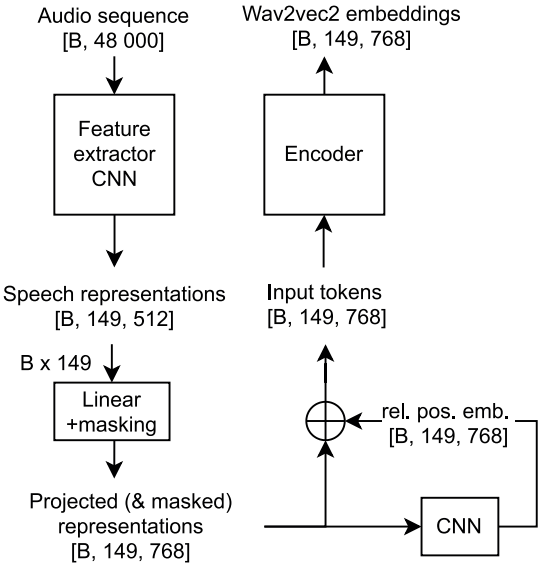


Figure 1: *Overview of the wav2vec 2.0 architecture. Shapes are specified for a batch of B audio samples with a length of 3 seconds.*

output of the feature extractor is an encoded vector sequence with dimensionality 512. Each vector has a receptive field of 20 ms which is similar to the window sizes in spectral-based representations.

Projection, SpecAugment & positional embedding

After the feature extraction LayerNorm (Lei Ba et al., 2016) is applied to each encoded vector representation in the sequence independently. After normalization, the representations are projected into 768 dimensions by a single, shared fully-connected layer, called the *feature projector*. On all projections dropout is applied (but no activation). Then, masking is applied over the whole sequence analogous to SpecAugment (Park et al., 2019); 0 or more random sets of consecutive vectors (masking in time domain) as well as 0 or more random sets of consecutive channels (masking in "frequency" domain) have their values blanked to 0. This masked projected sequence is then convolved by a single layer with a kernel size of 128, a stride of 1, padding of 64 and 16 groups followed by GELU activation in order to create a *relative positional embedding* for each projected representation. These relative positional embeddings are summed with the original input of the convolution, which changes the receptive field from 20 ms to 2.5 s. As a final step each vector is independently normalized with LayerNorm and dropout is applied again.

Transformer

The masked and projected sequence with both local and positional information is fed through a Transformer *encoder*. We only use the BASE variant, with 12 consecutive transformer layers. Each transformer layer consists of a residual 12-headed self-attention module and a residual 2-layer feed forward network with respectively 3072 and 768 hidden units. LayerDrop (Huang et al., 2016; Fan et al., 2020) is applied such that each transformer layer is potentially skipped. The final output sequence, with each representation potentially having both local and global information due to self-attention, is used in a downstream task.

2.2.2 Three wav2vec 2.0 variants for speaker recognition

The original wav2vec 2.0 framework fine-tunes on speech recognition by independently labeling each wav2vec 2.0 output embedding with a shared fully-connected layer, and optimizes with CTC loss (Graves et al., 2006). We propose two adaptions to this design for the speaker recognition task, inspired by BERT's (Devlin et al., 2019) single-sentence and sentence-pair classification setup.

Speaker recognition as single-utterance classification

The current paradigm in speaker recognition with deep neural networks is to train models with a classification-based approach (Snyder et al., 2018; Desplanques et al., 2020). To mimic this architectural paradigm two modifications to wav2vec 2.0 are made. First, the sequence of wav2vec 2.0 embeddings is reduced to a single embedding during training by pooling the output sequence. The pooling methods are described in Section 2.2.3. Second, we add a fully connected layer which uses the pooled embedding to classify each speaker in the training data with cross-entropy (CE) or angular additive softmax (AAM) loss (Deng et al., 2019; Liu et al., 2019). A speaker recognition trial is evaluated with the cosine similarity between two pooled embeddings. We refer to these variants as `w2v2-ce` and `w2v2-aam`.

Speaker recognition as utterance-pair classification

The second approach directly computes a (binary) similarity score. The two audio segments of a speaker recognition trial are first processed independently up to the encoder part of the network. Then, the two sequences of input tokens are concatenated, accompanied by special tokens at the embedding level: A start (a vector with all values $+1$), separator (a vector with all values -1), and end (also all values -1) token. The first wav2vec 2.0 embedding in the output sequence (corresponding to the start token) is used as input to a logistic regression with one dense layer. The singular output is the logit for the binary cross-entropy loss and the score for evaluating a speaker recognition trial. During training a batch consists of 8 speakers, 4 utterances per speaker, and 16 pairs each of same and different speakers. We refer to this third variant as `w2v2-bce`.

2.2.3 Pooling methods

We propose several pooling methods to reduce the variable-length sequence of wav2vec 2.0 embeddings to a fixed-size speaker embedding. We first consider the standard statistical pooling methods `mean`, `max`, `mean&std` and `quantile`. They aggregate each dimension over the time axis. The `mean&std` variant doubles the embedding dimensionality while `quantile` pooling expands each dimension five-fold with quantiles (0, 0.25, 0.5, 0.75, 1). We also assess taking the `first`, `middle` or `last` embedding of the sequence as a "pooling" strategy as well as selecting the index at `random`. Lastly we consider inserting a `start` token (i.e, a vector with all values +1) before the input sequence of the encoder and then selecting the first output token as the speaker embedding. Unlike BERT our `start` token does not have a meaningful prior due to the missing next-sentence prediction task during pre-training.

2.3 Experiments

2.3.1 Data

All experiments are conducted on the VoxCeleb1 (Nagrani et al., 2017) and VoxCeleb2 (Chung et al., 2018) datasets, which consist of interviews of celebrities extracted from YouTube. The VoxCeleb2 development set `vox2-dev`, which contains ~1.1 M audio files from 6 k speakers, is used for training. A validation set is created based on ~2 % of `vox2-dev`, which includes all speakers but does not overlap in recordings. For this validation set a random trial set of 5 k equal and 5 k different speaker pairs is generated, from which we compute the validation equal-error-rate (EER). This is used to select the best checkpoint during a training run as well as to tune hyperparameters. For evaluation we use the cleaned *original* (`vox1-o`, 40 speakers, ~37 k trials), *extended* (`vox1-e`, 1251 speakers, ~80 k trials) and *hard* (`vox1-h`, 1190 speakers, ~550 k trials, equal nationality and sex) test sets from VoxCeleb1 (Chung et al., 2018). There is no speaker overlap between the VoxCeleb1 test sets and the VoxCeleb2 dev set. The experiments with the wav2vec 2.0 network use the pre-trained weights[2] on Librispeech (Panayotov et al., 2015) released on HuggingFace (Wolf et al., 2020) by Fairseq (Ott et al., 2019).

2.3.2 Computational budget and fair comparison

We compare the performance of models under similar computational budgets. Each network is trained with a batch size of 3.2 M audio samples (Baevski et al., 2020a) by randomly sampling a chunk of 3 seconds from 66 different audio files. No data augmentation techniques are used. We train for 100 k iterations, approximately 6 epochs, with Adam (Kingma and Ba, 2015) and a OneCycle learning rate schedule (Smith and Topin, 2019). Given a (maximum) learning rate η, this learning rate schedule works as follows. First, in a warm up phase of 30 k iterations, the learning

[2]See https://huggingface.co/facebook/wav2vec2-base

rate gradually increases from $\frac{1}{25}\eta$ to η by cosine annealing. Then, for the remaining 70 k iterations, the learning rate gradually decreases from η to $\frac{1}{25000}\eta$, again with cosine annealing. We search for a well-performing learning rate η by first conducting a range test (Smith, 2017) which determines the range and step size for a grid search. The range test is done by performing 5 k training iterations with a linear learning rate schedule from $\eta = 0$ to $\eta = 1$. We observe three values from the resulting training loss curve, namely the learning rate η_{start} at which the loss initially starts decreasing, the learning rate η_{slope} where the loss has the steepest descent, and the learning rate η_{end} where the loss stops decreasing, plateaus or diverges. The grid search contains 7 learning rates, consisting of 4 logarithmic steps between η_{start} and η_{slope} as well as 4 logarithmic steps between η_{slope} and η_{end}. Models are evaluated with a cosine score between speaker embeddings lacking any further post-processing, except for the `w2v2-bce` variant which computes scores directly.

2.3.3 Baseline systems

We train two popular baseline models for speaker recognition, X-vector (Snyder et al., 2018) and ECAPA-TDNN (Desplanques et al., 2020) , implemented in SpeechBrain (Ravanelli et al., 2021), and compare them to the three wav2vec 2.0 adaptions `w2v2-ce`, `w2v2-aam` and `w2v2-bce`. All five models have the computation budget described in Section 2.3.2. The X-vector and ECAPA-TDNN networks use 40-dimensional filterbanks as input. The `w2v2-ce` and `w2v2-aam` variants use `mean&std` pooling which was chosen as it is also used in the x-vector network architecture. The `w2v2-aam` variant and ECAPA-TDNN use the AAM softmax loss with a scale of 30 and a margin of 0.2 in this and further experiments. The feature encoder part of the wav2vec 2.0 architecture is frozen for the whole training procedure following Baevski et al. (2020a).

2.3.4 Variation in pooling

Next we explore the different pooling methods proposed in Section 2.2.3. This was carried out for the single-utterance classification variants: `w2v2-ce` and `w2v2-aam`. We use the same training settings as in the baseline comparison and only vary the pooling method. Note therefore that the learning rate was tuned to the `mean&std` setup.

2.3.5 Ablation study

We perform several ablations on the best-performing wav2vec 2.0 variant for speaker recognition. These ablations are trained with 100 k iterations except in the experiments for the batch size. The first set of ablations study the effect of not freezing the feature extractor in the fine-tuning procedure as well as randomly initializing the whole network and thus not using any pre-trained weights. The second set of ablations explore the relative importance of the regularization techniques in the network architecture. We first disable only LayerDrop, then sequentially disable dropout and the masking of certain frames as well. The third set studies the effect of increasing or

Table 1: *The EER (in %) on three test sets of VoxCeleb1, for the baseline filterbank-based speaker recognition networks, and the proposed, fine-tuned wav2vec 2.0 variations. We conducted $N = 4$ training runs to compute the standard deviations.*

network	EER on `vox1-o`	EER on `vox1-e`	EER on `vox1-h`
X-vector	5.22 ± 0.12	5.60 ± 0.05	8.75 ± 0.05
ECAPA-TDNN	1.61 ± 0.03	1.69 ± 0.03	3.10 ± 0.05
`w2v2-ce`	2.25 ± 0.20	2.56 ± 0.10	4.91 ± 0.13
`w2v2-aam`	1.91 ± 0.12	2.22 ± 0.04	4.33 ± 0.08
`w2v2-bce`	7.28 ± 0.22	7.19 ± 0.22	11.34 ± 0.83

decreasing the chosen batch size by a factor of 2, but keeping the number of epochs constant. Thus, batch size 32 was trained for 200 k iterations while batch size 128 was trained for only 50 k iterations. The last ablations involve the learning rate schedule. We first test two constant learning rates: $3 \cdot 10^{-6}$ is the LR where the loss started increasing in the LR range test and 10^{-5} is the LR with the steepest decrease. The second schedule exponentially decays the LR from 10^{-5} to $3 \cdot 10^{-6}$. The final schedule is the tri-stage learning rate schedule, similar to Baevski et al. (2020a), which includes a warm-up phase linearly increasing the LR from 10^{-7} to 10^{-5} in 10 k iterations, a constant phase of 40 k iterations, and an exponentially decreasing stage from 10^{-5} to 10^{-7} for the remaining iterations.

2.4 Results

2.4.1 Baseline comparison

The LR range test and grid-tuning approach found the following learning rates: 10^{-4} for x-vector, 10^{-3} for ECAPA-TDNN, $9 \cdot 10^{-5}$ for `w2v2-ce`, $5 \cdot 10^{-5}$ for `w2v2-aam`, and $3 \cdot 10^{-5}$ for `w2v2-bce`. Table 1 shows four runs with these learning rates. We see that ECAPA-TDNN performs best on all test sets. The `w2v2-aam` network is the best performing wav2vec 2.0 variant. Both `w2v2-ce` and `w2v2-aam` improve on the x-vector architecture. Modeling speaker recognition as utterance-pair classification (`w2v2-bce`) performed worst.

2.4.2 Pooling methods

Table 2 compares the 9 different pooling strategies with the `w2v2-ce` and `w2v2-aam` networks. We observe that `start` pooling performs best for both networks. The difference between `start` pooling and the other pooling methods is more pronounced for `w2v2-aam` than for `w2v2-ce`. We also observe that the standard deviations for `w2v2-aam` are smaller compared to `w2v2-ce`. The low inter-model variance of random pooling shows that each wav2vec 2.0 embedding is a stand-alone speaker embedding. Although not shown, using random pooling in the evaluation for networks which were

Table 2: *The performance of different pooling strategies for the single utterance classification architectures on the* `vox1-e` *test set. Each method was trained with* $N = 3$ *random seeds. The evaluation of* `random` *pooling is shown for a single model* ($N = 1$)*, as well as averaged over* $N = 3$ *models. In both cases, the evaluation of the models is repeated four times.*

pooling method	EER with `w2v2-ce`	EER with `w2v2-aam`
`max`	4.79 ± 0.55	2.27 ± 0.04
`quantile`	2.75 ± 0.17	2.21 ± 0.03
`mean`	2.69 ± 0.06	2.11 ± 0.05
`mean&std`	2.60 ± 0.08	2.18 ± 0.03
`start`	2.52 ± 0.11	2.06 ± 0.03
`first`	2.61 ± 0.10	2.15 ± 0.05
`middle`	2.55 ± 0.07	2.18 ± 0.03
`last`	2.58 ± 0.07	2.18 ± 0.03
`random` ($N = 1$)	2.56 ± 0.00	2.40 ± 0.00
`random` ($N = 3$)	2.70 ± 0.13	2.37 ± 0.03

trained with `start` pooling degrades the EER with 0.2% absolute points. Moreover, creating ensembles out of different embeddings in the output sequence did not improve performance. This suggests that the transformer layer processes each embedding in the sequence similarly. We did observe a 0.2% absolute performance improvement when we averaged the last and penultimate layer of the encoder. This suggests that the speech representations at different layers potentially hold different information about the speaker in the utterance.

2.4.3 Ablation study

The results of the ablation study of `w2v2-aam` with `start` pooling are shown in Table 3. We see that unfreezing the feature extractor leads to better performance, but a frozen feature extractor is more stable across runs with different seeds. We also note a large degradation in performance when initializing with random weights instead of the pre-trained weights. We see that the regularization settings for fine-tuning on speech recognition are also beneficial for fine-tuning on speaker recognition, as disabling the regularization degrades the performance. Looking at the batch size, we see that increasing it beyond 3.2M audio samples does not increase performance, although it does decrease the variance slightly. Finally, we observe that using a learning rate schedule with a warm-up phase, such as the tri-stage or OneCycle schedule, is critical for stable training and optimal performance.

Table 3: *The performance under varying ablated configurations for the* `w2v2-aam` *network variant with* `start` *pooling. Each configuration was run with* $N = 3$ *random seeds. One run with exponential decay diverged and we therefore show* $N = 2$ *results for that row.*

ablation	EER on `vox-e`
default	2.06 ± 0.03
unfrozen feature extractor	1.88 ± 0.08
unfrozen feature extractor and random init	5.08 ± 0.08
no Layerdrop	2.45 ± 0.05
no LayerDrop, no dropout	2.39 ± 0.04
no LayerDrop, no dropout, no SpecAugment	2.39 ± 0.09
batch size 32	2.12 ± 0.10
batch size 128	2.05 ± 0.01
constant LR 10^{-5}	50.00 ± 0.00
constant LR $3 \cdot 10^{-6}$	3.71 ± 0.08
exponential decay ($N = 2$)	2.42 ± 0.07
tri-stage schedule	1.97 ± 0.04

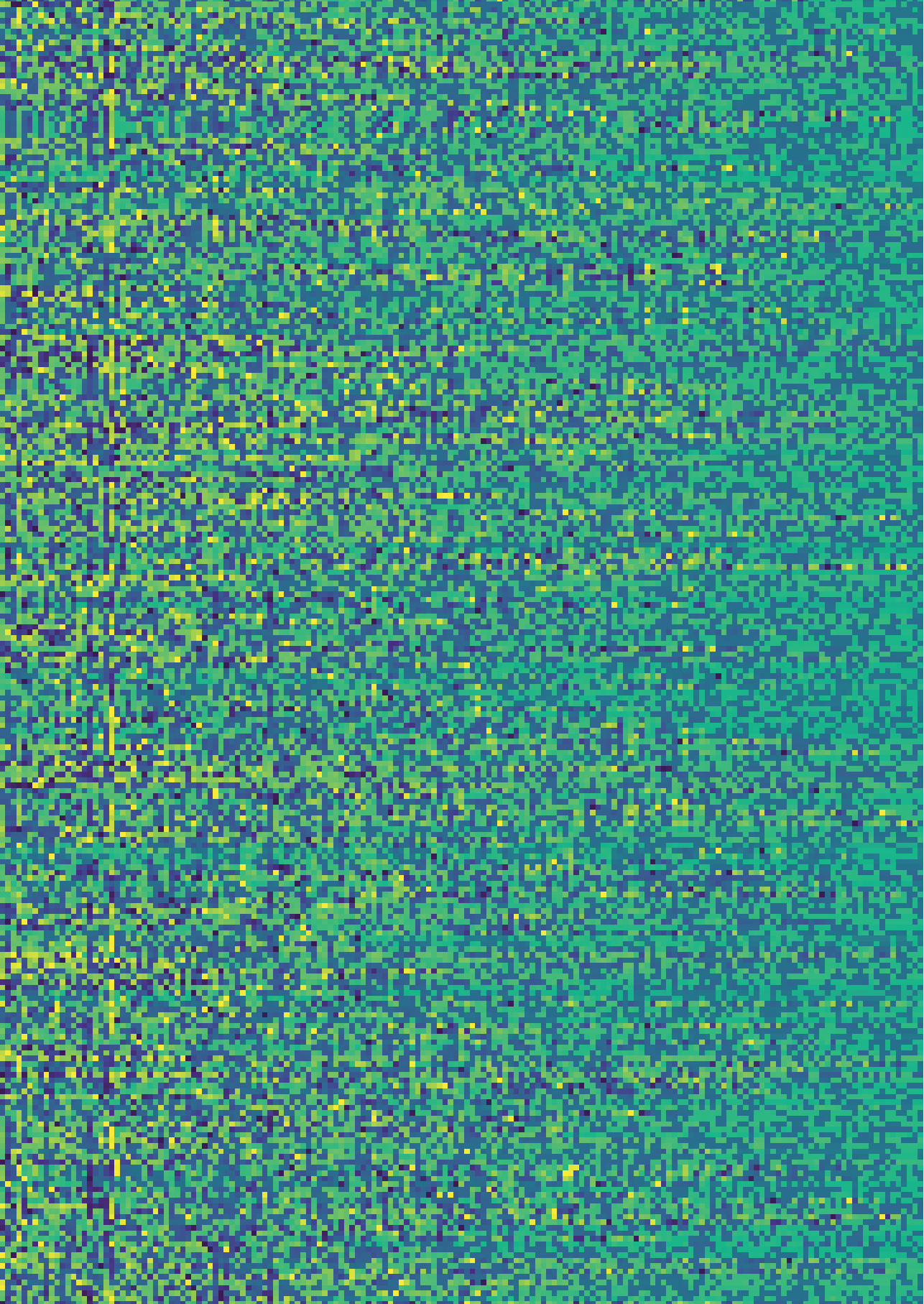
2.5 Conclusion and future work

We have shown that the wav2vec 2.0 framework can be successfully adapted to the speaker recognition task and that the pre-trained weights used for fine-tuning on speech recognition are also useful for fine-tuning on speaker recognition. All pooling methods we experimented with seemed to adequately replace the "class" token concept of the BERT model. However, fine-tuning wav2vec 2.0 did not result in better performance than the state-of-the-art ECAPA-TDNN. Good results with `start` pooling indicate that including a class token in the pre-training procedure is promising future work to improve speaker recognition performance, potentially improving on ECAPA-TDNN.[3]

Our proposed method `w2v2-bce`, which modeled speaker recognition as a paired-utterance classification problem, did not perform well. That said, it performed better than random chance, so the method does lead to speaker recognition capabilities. We hypothesize that optimizing with binary cross-entropy is more difficult compared to using multi-class cross-entropy, as the learning signal is significantly reduced. This reduction is due to the fact that multi-class cross-entropy enforces the network to

[3]Later work in the field (Chen et al., 2022a) has used the ECAPA-TDNN model *on top of* a self-supervised transformer network to achieve state-of-the-art results. In this chapter we simply use a linear layer as speaker recognition head.

create an embedding space in the last layer, over which 5994 distinct speakers can be classified using a single linear layer. For binary cross-entropy, there only need to be 2 clusters. The implicit bias to create 5994 clusters will benefit speaker recognition, as this many clusters can represent most variability between speakers, while 2 clusters cannot capture a lot of speaker variation. Future work could look into limiting the attention mechanism to the opposite utterance to simplify the learning task.



3 Training speaker recognition systems with limited data

In which our adventurer uncovers the capabilities of fine-tuning wav2vec 2.0 for speaker recognition in low-resource settings, and finds guidelines for collecting fine-tuning datasets in these limited data conditions.[4]

Recently, the wav2vec 2.0 framework (Baevski et al., 2020a) proposed a self-supervised pre-training, and consecutive fine-tuning approach for automatic speech recognition with a transformer network. Such a procedure has become the de facto standard in NLP with models like BERT (Devlin et al., 2019). One of the benefits of pre-training is the possibility to use large, unlabeled datasets, which are relatively inexpensive to obtain. Another benefit is that these pre-trained networks are flexible, and can be fine-tuned to a variety of related tasks. This has been shown to be the case for wav2vec 2.0 as well, which, while originally designed for speech recognition (Baevski et al., 2020a), has also been used for tasks like speaker recognition (Evain et al., 2021; Fan et al., 2021; Vaessen and van Leeuwen, 2022) and emotion recognition (Evain et al., 2021; Pepino et al., 2021; Yuan et al., 2021a). One property of fine-tuning a pre-trained network is that it requires less labeled data than training from scratch. For example, the authors of wav2vec 2.0 pre-train on 53 k hours of unlabeled speech data, fine-tune on 10 minutes of labeled speech data, and achieve a word-error-rate of 4.8 % on the clean test set of Librispeech (Panayotov et al., 2015). For comparison, in 2016 the DeepSpeech2 system (Amodei et al., 2016) achieved a 5.3 % word-error-rate with 3600 hours of labeled training data.

In this chapter, we want to study the behavior of wav2vec 2.0 under similar low-resource data conditions, but for speaker recognition instead of speech recognition. We are interested in the following research questions:

RQ 1. How well does the self-supervised, pre-trained wav2vec 2.0 network perform when fine-tuned for speaker recognition with only a small labeled dataset?

RQ 2. What is the most effective way to structure a small training dataset? Is there a trade-off to be made between speaker variability and session variability?

[4]This chapter is based on the publication Vaessen, N., and van Leeuwen, D. A. (2022). Training speaker recognition systems with limited data., in *Interspeech 2022*, 4760–4764. doi: 10.21437/Interspeech.2022-135.

We hypothesize that the pre-trained wav2vec 2.0 network will be effective in the low-resource speaker recognition setting, as the learned speech representations appear to be phonetic units, with high mutual information between these units and phonemes (Baevski et al., 2020a). This was shown useful as a basis for speech recognition, and it seems plausible that speaker recognition can benefit from these phonetic units of speech. Although we compare wav2vec 2.0 against non-self-supervised neural networks designed specifically for speaker recognition (Snyder et al., 2018; Desplanques et al., 2020) we speculate that these (or similar) networks can also benefit from self-supervised training. There has been work on self-supervised learning for speaker recognition (Thienpondt et al., 2020; Cai et al., 2021), and consecutive fine-tuning (Chen et al., 2022a), but to the extent of our knowledge, not for common speaker recognition networks (Snyder et al., 2018; Desplanques et al., 2020). Also, note that a frequent solution to limited data is data augmentation (Shorten and Khoshgoftaar, 2019). In this work, we explicitly skip data augmentation in order to observe the effects of self-supervised weights. The second research question is focused on data collection. There might be scenarios, related to, e.g., licensing, or the domain, where one needs to construct a dataset for fine-tuning. In this scenario, we hypothesize that maximizing the number of speakers in the dataset is the most effective strategy to achieve good fine-tuning performance.

3.1 Related work

Earlier work, such as Jayanna and Mahadeva Prasanna (2009), Das et al. (2014), and Poddar et al. (2018), interpret limited data availability not in the size of the training dataset, but in the length of the utterances. However, since the advent of neural approaches for speaker recognition, it has become standard practice to train with short audio segments, often between 0.5 and 3 seconds, as can be seen in, e.g., Snyder et al. (2018), Desplanques et al. (2020) and Lin and Mak (2020). In Wang et al. (2020) the contemporary field of few-shot learning is introduced, which considers low resource scenarios where (neural) models need to adapt to new classes ("N-way") with only a few samples ("K-shot"). In Li et al. (2020), the Librispeech dataset (Panayotov et al., 2015) is used to study low resource conditions for speaker identification. They vary the total training data length per speaker between 20, 40 or 60 segments of 3 seconds, and show only minor degradation in test accuracies when using either a prototypical loss (introduced by Snell et al. (2017)), or their proposed adversarial few-shot learning-based speaker identification framework. In Wang et al. (2019), speaker verification is considered within the few-shot learning paradigm with a subset of VoxCeleb2 (Chung et al., 2018), containing 71 train speakers and 30 test speakers. They compare the prototypical loss by Snell et al. (2017) against a triplet loss from Zhang and Koishida (2017), and train with 200 segments of 2 seconds. They show the prototypical loss achieves better equal-error-rates than the triplet loss in this scenario.

In the well-known series of NIST Speaker Recognition evaluations, the 2016 and 2018 editions (NIST, 2016; 2018) focused on language adaptation, which can be seen as a specific version of domain adaptation in a low-resource setting. For these evaluations, the test data contains speech in languages outside the background labeled training data, and some (i.e., low-resource) additional unlabeled training data in the evaluation languages is provided. Approaches include aligning the embedding distributions of source and target domain (Alam et al., 2018) and adding an adversarial loss in an end-to-end setting (Rohdin et al., 2019), where a critic is adversarially trained, along with the speaker recognition network, to discriminate between embeddings from source and target domain.

3.2 Methodology

3.2.1 Creating subsets of VoxCeleb2

In order to experiment with smaller dataset size conditions, we artificially limit ourselves to a subset of data from the so-called development set of VoxCeleb2 (Chung et al., 2018), which we use as *train* and *validation* set. This development set contains nearly 6 k speakers distributed over 1 M speech utterances, with a mean length of 7.8 seconds and a standard deviation of 5.2 seconds. Each speaker has a number of associated video recordings (sessions), and from each recording one or more speech utterances are automatically extracted by using face tracking, face verification, and active speaker verification. This ensures each speech utterance is attributed to a

Table 4: *Statistics on the* `vox2` *training split and the three low-resource subsets* `few-speakers`*,* `few-sessions` *and* `many-sessions` *we created for fine-tuning.*

statistic	`vox2`	`few-speakers`	`few-sessions`	`many-sessions`
duration (h)	2314	113	100	97
# speakers	5 994	100	5 994	5 994
# sessions	136 632	5 066	6 275	46 813
# utterances	1 068 871	49 400	47 952	47 952
sessions per speaker				
mean	23	51	1	8
min	4	22	1	4
max	89	87	4	8
utterances per session				
mean	8	10	8	1
min	1	1	1	1
max	264	264	8	3

single speaker, although some labeling noise is expected. Recordings were collected by Chung et al. (2018) from the top 100 results of a YouTube search. The search query included the name of the celebrity and the word "interview". We limit each subset to 50 k utterances, but vary the amount of speakers, the amount of sessions per speaker, or the amount of utterances per session. Throughout this chapter, we will refer to these datasets as `vox2`, `few-speakers`, `few-sessions`, and `many-sessions`. Statistics are shown in Table 4, and their creation is explained in more detail below.

The `vox2` entry refers to the train part of our train and validation split of the VoxCeleb2 development dataset. The validation split is used for monitoring overfitting, and selecting the best checkpoint. For each speaker in the original development set, we randomly move *sessions* to the validation split, until less than 99% of all utterances from the respective speaker are remaining. The validation split is created *before* choosing the utterances for the other subsets, i.e., `few-speakers`, `few-sessions` and `many-sessions` are subsets of the train split `vox2`. To calculate a validation EER, we create a random trial list with 15 k positive and negative (same-sex) trials. The validation set is equal for `vox2`, `few-sessions` and `many-sessions`. For `few-speakers`, we modify the validation set such that it only contains the 100 speakers in the subset, and a different random trial list, with 2 k positive and 2 k negative (same-sex) trials is created.

The `few-speakers` subset is designed to have relatively few speakers, but many sessions per speaker, and many utterances per session. The utterances in this subset are chosen by grouping the speakers from `vox2` by gender, and sorting descendingly by the amount of recordings available. We select the 50 female and male speakers with the highest number of recordings to be included in the subset.

The `few-sessions` subset is designed to have many speakers, but few sessions per speaker, and relatively many utterances per session. This subset contains all speakers in `vox2`. For each speaker, we sort their sessions descendingly by the number of utterances. We then select 8 utterances from each speaker. We start sampling from the session with the most utterances. If a session is exhausted (i.e., the session with the most utterances has less than 8 utterances), we continue with the next session according to the sorted collection, and so forth, until 8 utterances are collected.

The `many-sessions` subset, in contrast with `few-sessions`, is designed to have many speakers, relatively many sessions per speaker, but few utterances per session. Just as in `few-sessions`, we select 8 utterances from all speakers in `vox2`. However, the difference is that we sample only one utterance per session. When a speaker has fewer than 8 sessions available, we cycle through the sorted collection, selecting only one utterance per session per cycle, until 8 utterances are selected.

3.2.2 Speaker recognition networks

We train three different speaker recognition models on the datasets in Section 3.2.1. We use third-party library network implementations, but train and evaluate with our own PyTorch (Paszke et al., 2019) code.

X-vector

The X-vector architecture (Snyder et al., 2018) is a popular neural network for speaker recognition and diarization, initially from the Kaldi framework (Povey et al., 2011). The X-vector network consists of 5 consecutive layers with 1-d convolutions, ReLU activation, and BatchNorm, followed by *mean and standard deviation* pooling, and two fully-connected (FC) layers. During training, a third FC layer is used for computing the classification loss. We extract the speaker embeddings from the first FC layer. We use the implementation by SpeechBrain (Ravanelli et al., 2021), with the default settings, such that the speaker embeddings have a dimensionality of 1024.

ECAPA-TDNN

The ECAPA-TDNN architecture (Desplanques et al., 2020) is a more recent speaker recognition network that showed best performance in the VoxCeleb 2020 challenge (Nagrani et al., 2020). It builds on top of the X-vector paradigm by adding global context through network architecture modifications. First, it makes use of three consecutive res2blocks (Gao et al., 2021), consisting of three 1-d convolutions, a squeeze-and-excitation layer (Hu et al., 2018) and a skip connection (He et al., 2016). They also aggregate the output of each res2block, before using channel-wise attentive statistical pooling to compute a fixed-size speaker embedding. We use the SpeechBrain (Ravanelli et al., 2021) implementation, with the default settings, such that the speaker embeddings have a dimensionality of 128.

Wav2vec 2.0

The wav2vec 2.0 architecture applies self-supervised pre-training to speech data, and has been used for multiple speech-related tasks. We only fine-tune the network. There is a BASE and LARGE variant, we only consider the BASE network. The architecture consists of three components. First, raw audio is processed by a 7-layer CNN with 1-d convolutions, LayerNorm (Lei Ba et al., 2016), and GELU activation (Hendrycks and Gimpel, 2016). Secondly, a linear projection, consisting of a LayerNorm and a single FC layer, is applied. Then, an additive relative positional embedding is added with the use of a 1-layer CNN. Lastly, the hidden state sequence is processed by 12 transformer blocks. We use the Transformers (Wolf et al., 2020) implementation, with self-supervised weights provided by Fairseq[5]. The self-supervision was carried out (by Fairseq (Baevski et al., 2020a)) on the Librispeech (Panayotov et al., 2015) dataset.

[5]We downloaded the weights from https://huggingface.co/facebook/wav2vec2-base

For the speaker recognition task, the final hidden state sequence is mean-pooled into a fixed-size speaker embedding of dimensionality 768 (Fan et al., 2021; Vaessen and van Leeuwen, 2022). During training, a single FC layer is used for classification.

3.3 Experiments

The experiments consist of finding the best learning rate for each network and dataset combination (Section 3.3.2), multiple runs with the best learning rate while varying the amount of training iterations (Section 3.3.3), and an ablation study (Section 3.3.4).

3.3.1 Training and evaluation protocol

We base the following training protocol on the ECAPA-TDNN (Desplanques et al., 2020) and wav2vec 2.0 (Baevski et al., 2020a) articles. Each network is trained for n_{steps} steps with the Adam (Kingma and Ba, 2015) optimizer. We use a cyclic learning rate (LR) schedule (Smith, 2017) with a decaying maximum LR according to the *triangular2* policy. We always use 4 cycles, one cycle is therefore $n_{\text{steps}} / 4$ iterations. The minimum LR each cycle is 10^{-8}. For the `vox2` dataset we validate every 5 k steps, for the low-resource datasets we validate after each epoch. To create a batch, we shuffle the order of the utterances each epoch, select 100 utterances, and take a random 2 second chunk from each utterance. This matches the total batch size of 3.2 M audio samples used in Baevski et al. (2020a) and Vaessen and van Leeuwen (2022). For the X-vector and ECAPA-TDNN network we input a 80-dimensional MFCC with a window length of 25 ms and a 12.5 ms shift. All three network are trained with angular additive margin softmax loss (Deng et al., 2019; Liu et al., 2019). We use a margin of 0.2 and a scale of 30. We do not use any weight decay in order to reduce the search space. For X-vector and ECAPA-TDNN, we use SpecAugment (Park et al., 2019) with 5 to 10 masks of length 10 in the time axis, and 1 to 3 masks of length 4 in the channel axis. For wav2vec 2.0, we use LayerDrop (Huang et al., 2016; Fan et al., 2020) of 10% in the transformer layers, and dropout of 10% is applied after each fully-connected layer in the network. The wav2vec 2.0 network also includes masking before the relative positional embedding is added, similar to SpecAugment; 10% of the channel dimensions are randomly masked, and 50% of the time steps are randomly masked. We freeze the whole wav2vec 2.0 network for the first 12.5 k steps, except for the last FC layer used for the logits of the speaker classification task. We also freeze the feature extractor CNN for the whole training run (n_{steps} iterations). Training is conducted on a RTX 3090 GPU for wav2vec 2.0, and a GTX 2080Ti GPU for X-vector and ECAPA-TDNN. All experiments are capped to 32 GB RAM and 6 CPU cores. In total 209 days of GPU time was used for experiments.

We evaluate trials using a cosine score between speaker embeddings, without any other processing. We use the original VoxCeleb1 (Nagrani et al., 2017) test set (40 speakers, henceforth `vox1-o`) as a development set, and the VoxCeleb1 hard test

set (1190 speakers, henceforth `vox1-h`) as the evaluation set. There is no overlap between the VoxCeleb1 (Nagrani et al., 2017) dataset, used for evaluation, and the VoxCeleb2 (Chung et al., 2018) dataset, used as source for our fine-tuning data. There *is* an overlap between the development set `vox1-o` and evaluation set `vox1-h`, but we verified that the results are not significantly different when the trials from overlapping speakers are removed.

3.3.2 Learning rate search

We conduct a learning rate search for the maximum LR in the cyclic schedule. This is done for all three networks and all four datasets. We conduct this search in two phases. In the first phase we scan over a large magnitude: we consider 10^{-i}, with $i \in \{2, 3, 4, 5, 6, 7\}$. Based on the development set, we select the LR 10^{-j} with the lowest EER. In the second phase, we scan around this LR: we consider $\{1.78, 3.16, 5.62\} \times \{10^{-j-1}, 10^{-j}\}$. After the second phase, the LR with the lowest EER is used for the remaining experiments. The random seed is kept constant across all training runs in the grid search, and thus every learning rate is attempted only once. Each run has $n_{\text{steps}} = 50\,\text{k}$.

The best LR, and the respective EER on the development set `vox1-o`, are shown in Table 5. We see that wav2vec 2.0 performs best on all datasets. However, performance on `few-sessions` is suboptimal for all three networks. In Figure 2 we plot the learning rate against the EER. In general, we can see that wav2vec 2.0 requires a lower learning rate. Moreover, for all three networks, the optimal learning rate is dependent on the dataset.

3.3.3 Varying n_{steps} with optimal learning rate

In the following experiments, we vary n_{steps} to 25 k, 50 k, 100 k, and 400 k for each network and dataset combination. Additionally, we run each experiment with 3 different random seeds, and we use the optimal LR found in the learning rate search. The networks are evaluated on the `vox1-h` evaluation set.

The results are shown in Table 6. First, we see that ECAPA-TDNN has the best performance on `vox2`, while wav2vec 2.0 is slightly worse than ECAPA-TDNN but better than the X-vector network. We observe the same on the `few-sessions` dataset, although the EER values are much higher compared to the other three datasets. On `few-speakers` and `many-sessions`, we see that wav2vec 2.0 has the best performance, while ECAPA-TDNN is slightly worse, but better than the X-vector network. Moreover, we see that on all three low-resource datasets the wav2vec 2.0 network achieved the best performance with 50 k steps, while ECAPA-TDNN and X-vector almost always have the best performance after 400 k steps.

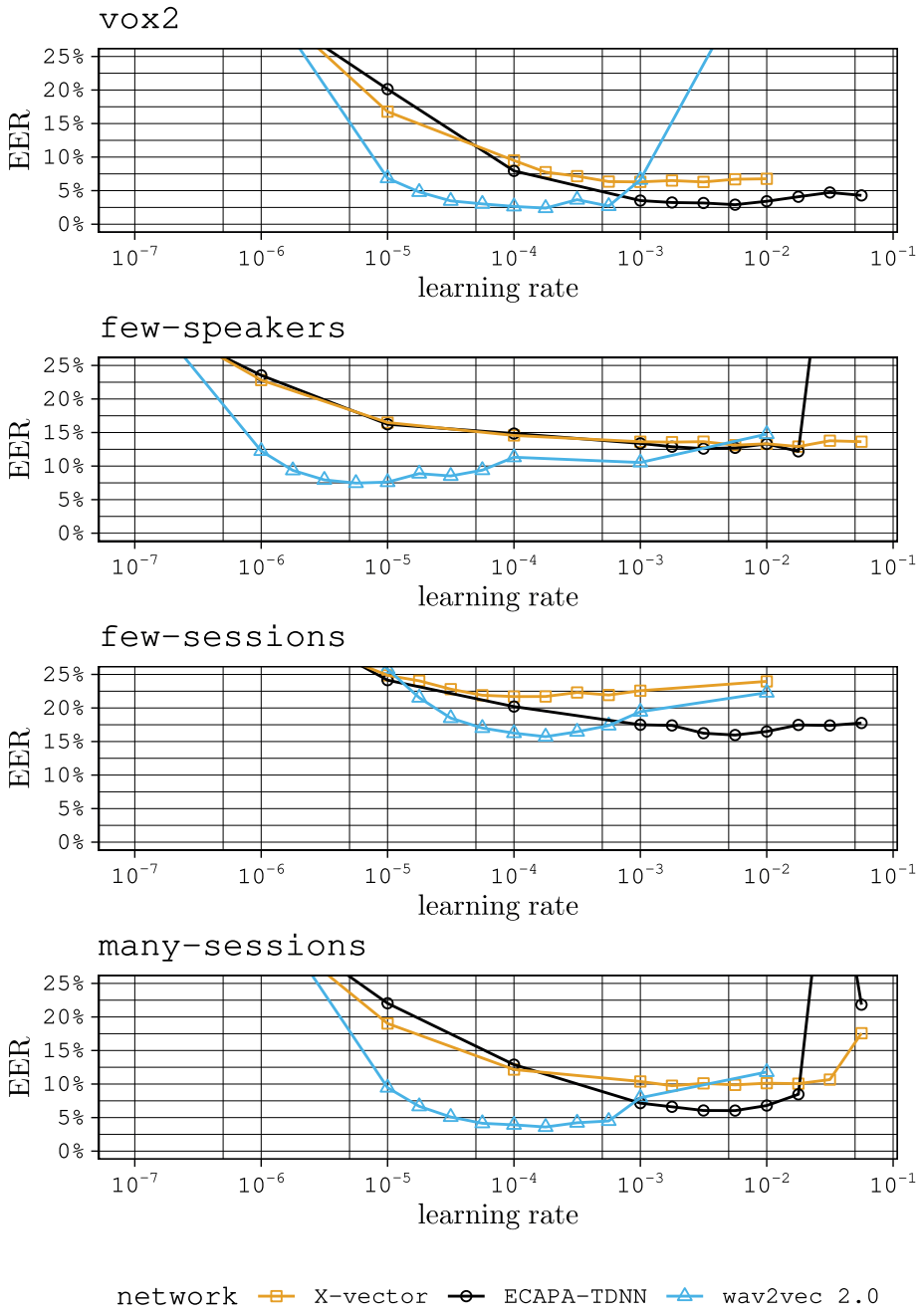


Figure 2: *The results of the learning rate search. We plot the learning rates of phase 1 and phase 2 of the grid search on the x-axis, and the EER on the* `vox1-o` *development set on the y-axis. Each sample represents a single experiment, no variation is measured.*

Table 5: *The best-performing learning rate for each network and dataset combination, trained for 50k steps. The EER is measured on the* `vox1-o` *development set.*

data		X-vector	ECAPA-TDNN	wav2vec 2.0
`vox2`	LR	$3.16 \cdot 10^{-3}$	$5.62 \cdot 10^{-3}$	$1.78 \cdot 10^{-4}$
	EER	6.30 %	2.91 %	2.40 %
`few-speakers`	LR	$1.78 \cdot 10^{-2}$	$1.78 \cdot 10^{-2}$	$5.62 \cdot 10^{-6}$
	EER	12.91 %	12.19 %	7.46 %
`few-sessions`	LR	$1.00 \cdot 10^{-4}$	$5.62 \cdot 10^{-3}$	$1.78 \cdot 10^{-4}$
	EER	21.70 %	15.97 %	15.72 %
`many-sessions`	LR	$1.78 \cdot 10^{-3}$	$5.62 \cdot 10^{-3}$	$1.78 \cdot 10^{-4}$
	EER	9.75 %	6.04 %	3.60 %

3.3.4 Ablation study

For the last set of experiments we perform an ablation study on the baseline training protocol described in Section 3.3.1. We perform the ablations on the `few-speakers` and `many-sessions` datasets with the wav2vec 2.0 network. All training runs are done with $n_{\text{steps}} = 50\,\text{k}$, and the best LR found in the grid search.

In the first set of ablations, we are interested in the importance of the cycling learning rate schedule. To observe this, we test the following variations in the learning rate schedule:

1. using a constant learning rate instead of a cyclic schedule, where the constant learning rate is equal to the maximum learning rate of the original schedule.
2. an exponentially decaying schedule, i.e., no warm-up effect, where we start with the maximum learning rate of the original schedule and decrease it in n_{steps} to the minimum learning rate of the original schedule.
3. the original cyclic schedule but modified to have only one cycle instead of four cycles, meaning 25 k steps up and 25 k steps down.

In the second set of ablations, we focus on the pre-trained weights and how they are updated. We use the following variations:

1. randomly initializing the wav2vec 2.0 network.
2. starting with the self-supervised weights, but without freezing any layers at any point during fine-tuning.
3. starting with the self-supervised weights, and freezing the feature extractor CNN for 50 k steps, i.e., for the whole fine-tuning procedure.
4. starting with the self-supervised weights, and freezing the whole network, except for the classification FC layer, for the first learning rate cycle (12.5 k steps). For the remaining three cycles (37.5 k steps) all weights are updated.

Table 6: *Each network and dataset combination is trained for 25k, 50k, 100k, and 400k steps with the learning rate from Table 5. The EER values are measured on the* `vox1-h` *evaluation set. Each experiment was run 3 times.*

steps	X-vector	ECAPA-TDNN	wav2vec 2.0
`vox2`			
25k	16.30 ± 0.64	6.80 ± 0.06	7.76 ± 0.07
50k	11.21 ± 0.33	5.46 ± 0.07	4.66 ± 0.15
100k	7.21 ± 0.10	4.61 ± 0.13	4.20 ± 0.16
400k	5.01 ± 0.04	3.93 ± 0.07	5.90 ± 0.77
`few-speakers`			
25k	18.93 ± 0.15	18.27 ± 0.03	22.48 ± 0.16
50k	18.02 ± 0.08	17.59 ± 0.40	15.19 ± 0.24
100k	18.00 ± 0.22	17.48 ± 0.07	15.60 ± 0.20
400k	18.04 ± 0.31	16.95 ± 0.15	18.50 ± 0.43
`few-sessions`			
25k	28.46 ± 0.12	22.67 ± 0.23	23.78 ± 0.10
50k	27.23 ± 0.14	21.25 ± 0.44	21.88 ± 0.16
100k	26.29 ± 0.30	20.58 ± 0.25	22.08 ± 0.29
400k	24.00 ± 0.19	19.52 ± 0.12	23.31 ± 0.10
`many-sessions`			
25k	18.00 ± 0.11	11.51 ± 0.39	10.41 ± 0.52
50k	16.05 ± 0.31	9.78 ± 0.06	6.72 ± 0.04
100k	13.53 ± 0.73	9.12 ± 0.11	7.66 ± 0.83
400k	10.58 ± 0.25	9.36 ± 0.08	7.93 ± 0.37

The third set of ablations consider regularization techniques applied during fine-tuning. We test the following variants:

1. disabling all regularization parameters (so no dropout, LayerDrop, nor masking).
2. only enabling dropout.
3. only enabling LayerDrop.
4. only enabling masking.

The results of the ablation study are shown in Table 7. When we ablate on the learning rate schedule, we observe that for `few-speakers` all three schedules perform worse than the baseline. For `many-sessions`, an exponentially decaying schedule seems to perform slightly better than our baseline, while a constant schedule, as well as a cyclic schedule with one cycle, perform worse. Next, we looked at the network weights and

Table 7: *Ablation on the wav2vec 2.0 network trained on the* `few-speakers` *and* `many-sessions` *datasets. Evaluation is done on the* `vox1-h` *evaluation set. Each experiment is run 3 times.*

ablation	`few-speakers`	`many-sessions`
baseline (Table 6)	15.19 ± 0.24	6.72 ± 0.04
LR schedule		
constant	16.77 ± 0.26	8.80 ± 0.44
exponential decay	16.68 ± 0.20	6.67 ± 0.04
1 cycle	15.79 ± 0.23	8.59 ± 0.23
weights		
random init	33.35 ± 0.16	46.24 ± 0.06
pre-trained (nothing frozen)	15.16 ± 0.17	7.18 ± 0.19
pre-trained + CNN frozen	15.48 ± 0.25	7.83 ± 0.45
pre-trained + frozen 1st cycle	14.52 ± 0.11	6.38 ± 0.17
regularisation		
none	16.67 ± 0.27	7.73 ± 0.07
only dropout	16.67 ± 0.11	8.01 ± 0.12
only LayerDrop	15.03 ± 0.14	6.72 ± 0.21
only masking	16.21 ± 0.18	7.87 ± 0.24

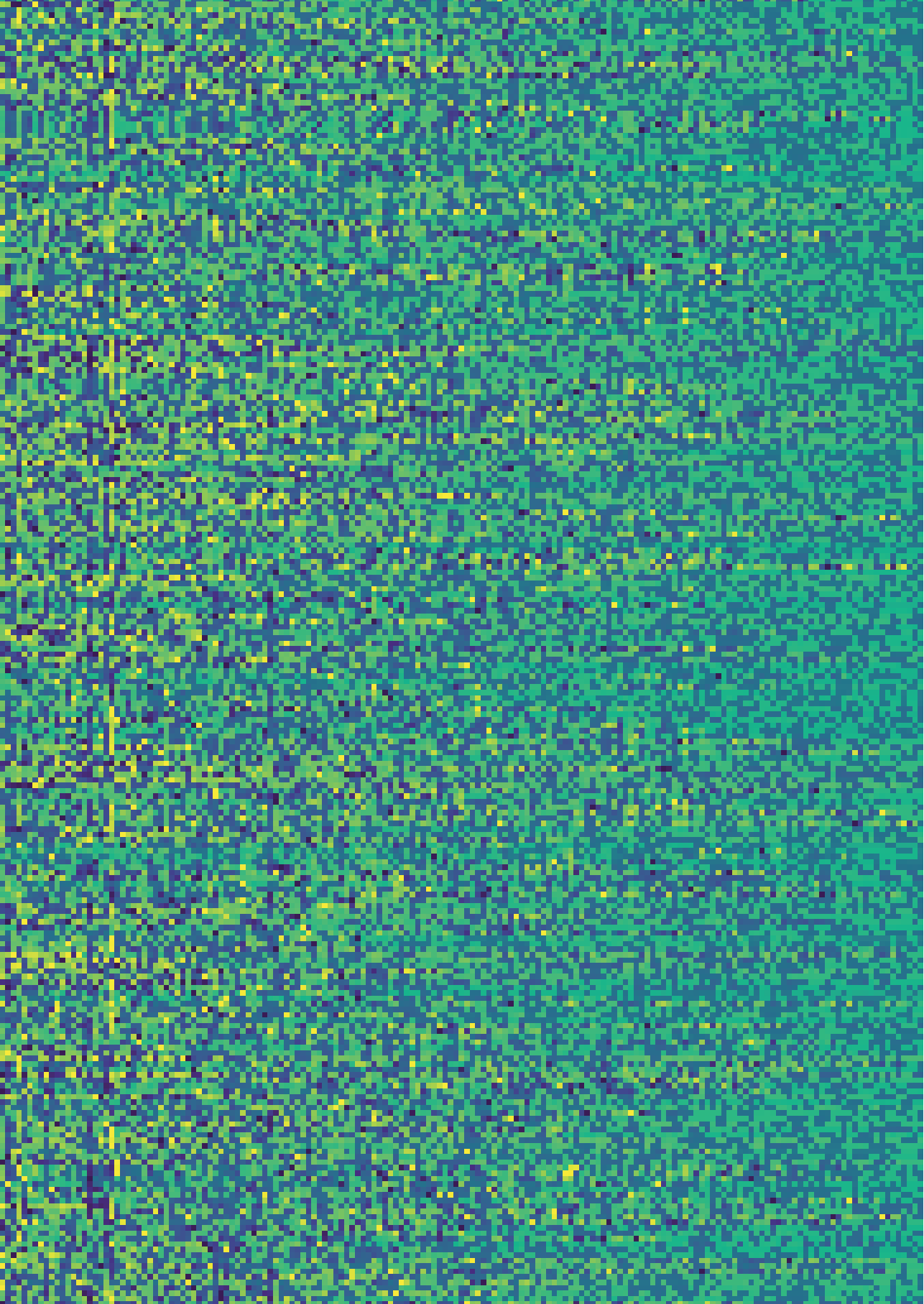
the freezing schedule. We observe that using randomly initialized weights prevents convergence to a reasonable performance. When using the self-supervised pre-trained weights without any freezing, the performance is slightly worse than the baseline. This is also the case for freezing the feature extractor CNN for n_{steps}. However, freezing the whole network for the first learning rate cycle seemed beneficial, as it improves on the baseline. This is due to the fact that, compared to the baseline, the feature extractor CNN can be updated for the last three cycles. Finally, we observe that disabling all regularization degrades performance. With only enabling LayerDrop regularization we achieve similar performance to the baseline, while only enabling masking, and only enabling dropout, perform similar to disabling regularization.

3.4 Conclusion

We have shown that the wav2vec 2.0 network, when initialized with self-supervised weights, has better performance, and needs fewer training iterations, than the X-vector and ECAPA-TDNN network on two out of the three low-resource fine-tuning datasets. Although ECAPA-TDNN performed slightly better on the `few-sessions` dataset, all three networks demonstrated suboptimal performance. As indicated by

the results on `few-sessions`, a dataset with many speakers but no session variability leads to poor performance. As all networks had better performance on `few-speakers` compared to `few-sessions`, it seems that a low-resource speaker recognition dataset should have fewer speakers with more sessions per speaker. However, `few-speakers` has a mean of 51 sessions per speaker, compared to a mean of 8 for `many-sessions`, while achieving worse EERs. It would be interesting future work to find an optimal balance between the amount of speakers and the amount of sessions per speaker.

Currently, the self-supervised learning optimization of wav2vec 2.0 uses a contrastive loss to distinguish a masked segment from other segments in the same utterance. For speaker recognition, it might be beneficial to include segments from other utterances, which could allow for modelling inter- and intra-speaker variance. Such a change could then perhaps result in even better performance when fine-tuning with low-resource datasets, and specifically with `few-sessions`. We are also interested in future work on pre-training with a dataset other than Librispeech, which has limited variability per speaker. The pre-training might be more effective for speaker recognition if the dataset has more session variability. It might also be relevant to pre-train on VoxCeleb2, so that the model has been fine-tuned on in-domain data, which might lead to better fine-tuning performance while requiring less labeled data.



4 Towards multi-task learning of speech and speaker recognition

In which our adventurer uncovers the difficulty of training a model capable of simultaneous speech and speaker recognition.[6]

Speech and speaker recognition are, in a sense, orthogonal speech technology tasks. When we develop automatic speech recognition (ASR) systems, a very desirable property is speaker independence: we want the system to perform well irrespective of who uttered the words. Therefore, neural ASR models should learn to generate speech embeddings which have minimal variability when the same text is spoken by different speakers. In contrast, when developing speaker recognition (SKR) systems, a very desirable property is text independence: we want the system to perform well irrespective of what was said. Neural SKR models, then, should learn to generate speaker embeddings which have minimal variability when the same speaker utters different texts. We observe a dichotomy where ASR models should be invariant to who speaks while SKR models should be invariant to what is being said. This raises the question: is it possible to train a multi-task learning (MTL) model which can do both speaker and speech recognition, while the ASR and SKR components respectively need to be invariant to who is speaking, and what is said?

Besides this interesting academic question, fully-fledged ASR applications often involve speaker recognition components, in order to provide, e.g., speaker-attributed transcriptions, or speaker diarization. Moreover, previous work has used speaker information to improve the performance of ASR models, in the classical hybrid artificial neural network and hidden Markov model approach (BenZeghiba and Bourlard, 2003), and in more recent recurrent neural network models (Peddinti et al., 2015). However, in these cases, there are still two *explicit* models for speech and speaker information, which are combined ad hoc to achieve performance gains. We are interested in a network *implicitly* learning speech and speaker representations,

[6]This chapter is based on the publication Vaessen, N., and van Leeuwen, D. A. (2023). Towards Multi-task Learning of Speech and Speaker Recognition., in *INTERSPEECH 2023*, 4898–4902. doi: 10.21437/Interspeech.2023-353.

and internally making use of these representations to achieve the same gains. As an added benefit, this could reduce the complexity of fully-fledged ASR applications to a single neural network model.

However, we observe the following obstacles in bringing these tasks together:

1. Differences in neural architectures for respective tasks, although transformers are bridging this gap.
2. Datasets for ASR lack session variability, while datasets for SKR lack transcriptions.
3. ASR training must be carried out on complete utterances. Typically, ASR datasets do not have time-aligned transcriptions at the word level, while SKR network training is done on short segments as training on long utterances prevents generalization.

We choose to build on top of the wav2vec 2.0 framework, as the same architecture has been fine-tuned in a single-task learning (STL) setting to both ASR (Baevski et al., 2020a), and SKR (Fan et al., 2021; Vaessen and van Leeuwen, 2022), bridging the gap between neural architectures for ASR and SKR. Our proposed multi-task model is trained with Librispeech (Panayotov et al., 2015) for ASR and VoxCeleb2 (Chung et al., 2018) for SKR. We train with disjoint steps, meaning batches only contain data from one of the two datasets. This enables ASR training on complete utterances and SKR training on short segments. This allows us to answer the following research questions:

RQ. 1 Can a transformer-based architecture perform ASR and SKR simultaneously?
RQ. 2 Is it feasible to train an MTL model with state-of-the-art datasets for speech recognition and speaker recognition?
RQ. 3 Can we train with the complete sentence as input for ASR while using short segments as input for SKR?

As wav2vec 2.0 has been shown to work for both the ASR and SKR task, we hypothesize that the transformer based architecture can learn embeddings which contain both speech and speaker information. Prior work (Vaessen, 2020) on disjoint multi-task learning for object detection and segmentation has shown feasibility of disjoint optimization. Therefore, we expect this training strategy to also work for ASR and SKR. Finally, we hypothesize that the input length difference for ASR and SKR is solvable due to the fact that wav2vec 2.0 naturally uses variable-length input, during training and inference for ASR, and for inference for SKR as well.

4.1 Background

4.1.1 Related MTL work

Fan et al. (2021) use the wav2vec 2.0 network for multi-task learning between the speech technology tasks of speaker recognition and language identification. Their MTL model did not improve on baseline STL performances. Adi et al. (2019)

consider whether ASR systems can benefit from MTL learning of speaker recognition, or whether adversarial learning (AL) as proposed by Ganin et al. (2016) is more beneficial. Using the WSJ dataset by Paul and Baker (1992), and a CNN model, they find similar, but small, improvement gains with MTL and AL. Also, Tang et al. (2016) train a MTL speech and speaker recognition network on WSJ. They use two interconnected LSTMs, one for each task. The output of each LSTM is shared in the next time step. Concurrently, Pironkov et al. (2016) train an LSTM for ASR, with SKR as auxiliary task, on the TIMIT dataset by Garofolo (1993). Lastly, the recent Whisper model by Radford et al. (2023) is a multi-task transformer model with impressive ASR performance, which is also capable of doing speech activity detection, language identification and speech translation. However, the absence of the speaker recognition task in Whisper is notable, and indicates that more work on combining speech and speaker recognition is desirable.

4.1.2 Wav2vec 2.0

An important aspect of the wav2vec 2.0 framework (Baevski et al., 2020a) is the application of self-supervised learning to initialize the network weights based on unlabeled data, before fine-tuning the network on (a smaller amount of) labeled data. In this work, we limit ourselves to fine-tuning the network in a multi-task configuration. Further details on the self-supervised learning aspect can be found in the seminal work (Baevski et al., 2020a).

The wav2vec 2.0 architecture consists of three components. First, a 1-d feature extractor CNN processes a raw audio waveform $\mathcal{X} = x_1, \ldots, x_n$ into frames of speech features $\mathcal{Z} = z_1, \ldots, z_m$, with a window size of 20 ms. These features are projected, potentially masked in the time and feature dimension to mimic SpecAugment (Park et al., 2019) regularization, and a relative positional embedding is added. The resulting sequence of input vectors, with a receptive field of 2.5 s, are processed by an encoder network (Devlin et al., 2019) with multi-head attention transformer layers (Vaswani et al., 2017) to produce a sequence of output vectors $\mathcal{C}^L = c_1^L, \ldots, c_m^L$, where L specifies the output sequence of a specific transformer layer. The output sequence (of any layer, but usually the last one) can be used by a downstream task.

For ASR, the output vectors of the wav2vec 2.0 network can represent phones or letters. A single fully-connected (FC) layer can be used to classify each vector, and with CTC loss (Graves et al., 2006) the network is trained end-to-end. For SKR, the output vectors are pooled into a fixed-length speaker embedding (Fan et al., 2021; Vaessen and van Leeuwen, 2022). The network is trained end-to-end by classifying speaker identities using the speaker embedding and a single FC layer.

4.2 Methodology

4.2.1 MTL network architectures

Two task-specific heads

Throughout the work we only use the `BASE` wav2vec 2.0 network architecture with 12 transformer layers. We only make slight modifications for our multi-task purposes by adding two task-specific heads; one for speech recognition, and one for speaker recognition. The automatic speech recognition head consists of a single FC layer which predicts a softmax probability distribution over the vocabulary, for each wav2vec 2.0 output token in the sequence $\mathcal{C}^{12} = c_1^{12}, \ldots, c_m^{12}$. This is equivalent to the original ASR design (Baevski et al., 2020a).

The speaker recognition head consists of two components. The first part transforms the output sequence into a speaker embedding by using a pooling method. We will detail these pooling methods in more detail below. The second part, only used during training, is a single FC layer used to classify the speaker identity based on the pooled speaker embedding.

We consider both heads using $\mathcal{C}^{12}$ as input, which implies $\mathcal{C}^{12}$ contains speaker and speech information. However, we also experiment with using $\mathcal{C}^n$ as input for the speaker head instead. In this configuration, the network can gradually remove speaker information from $\mathcal{C}^{n+1}$ onward. We chose layer $n = 6$ so that half of the transformer network can be solely focused on speech recognition.

Speaker embeddings

We compare three strategies to extract a speaker embedding from an output sequence $\mathcal{C}^n$. The first, `mean` pooling, simply aggregates each feature dimension of the wav2vec 2.0 output vectors $c_1^n, \ldots, c_m^n$ over the time-axis (Fan et al., 2021) by taking the average. The second, `first` pooling (Vaessen and van Leeuwen, 2022), does not consider the actual output sequence. Instead, we simply use the first token c_1^n as a speaker embedding. The third variant `ecapa` uses the ECAPA-TDNN (Desplanques et al., 2020) architecture to compute a speaker embedding, with the sequence $\mathcal{C}^n$ as input to ECAPA-TDNN, similar to WavLM (Chen et al., 2022a). Note that by using `mean` or `ecapa` pooling, there needs to be speaker information throughout the output sequence, while for `first` pooling the speech and speaker information can be separated by the transformer layers.

4.2.2 Optimization

We want to train the network on state-of-the-art datasets for speaker and speech recognition. In this section, we suggest two methods for MTL optimization for these tasks. They are based on using Librispeech (Panayotov et al., 2015), a well-known dataset for speech recognition, and VoxCeleb2 (Chung et al., 2018), a well-known

speaker recognition dataset. We chose to include VoxCeleb2 as Librispeech is not well-suited for speaker recognition. This is because Librispeech has relatively few speakers (2484 versus 5994), and little session variation. This low variation is due to the fact that most speakers in the Librispeech dataset record the whole book in a single acoustic environment. In contrast, VoxCeleb2 has high variation in speakers, as it includes interviews of celebrities in different settings, including, for example, a TV studio, a football stadium, or a red carpet at a formal event. However, VoxCeleb2 does not have transcriptions, which complicates multi-task learning. We suggest a joint optimization strategy, where we make use of ASR pseudo-labels for VoxCeleb2, and a disjoint optimization strategy, where we have two batches and forward steps for each optimization step, with each batch containing either speaker or speech labels.

Disjoint training

In order to train with Librispeech and VoxCeleb2, we propose to optimize our network with a disjoint forward step. We assume the availability of two data sources,[7] namely a speech dataset D_s and a speaker dataset D_k, as well as a base network weights θ_b, speech head weights θ_s, and speaker head weights θ_k. We also have a base network function N, a speech recognition head function H_s with loss function L_s, as well as a speaker recognition head function H_k with loss function L_k.

Each iteration i, we sample a speech batch $(x_s^i, y_s^i) \in D_s$ and a speaker batch $(x_k^i, y_k^i) \in D_k$. We then apply two forward passes, one on the speech batch and one on the speaker batch, where we write $p \in \{s, k\}$:

$$
\begin{aligned}
q_p^i &= N(x_p^i, \theta_b^i) \\
\hat{y}_p^i &= H_p(q_p^i, \theta_p^i) \\
L_p^i &= L_p(y_p, \hat{y}_p^i)
\end{aligned}
$$

The total loss L^i is a weighted sum over the speech and speaker loss:

$$
L^i = \lambda_s L_s^i + \lambda_k L_k^i
$$

with $\lambda_{\{s,k\}}$ the weights for speech and speaker and $\lambda_s + \lambda_k = 1$. The gradients for the different parts of the network become:

$$
\begin{aligned}
\nabla_{\theta_k} L^i &= \lambda_k \nabla_{\theta_k} L_k^i \\
\nabla_{\theta_s} L^i &= \lambda_s \nabla_{\theta_s} L_s^i \\
\nabla_{\theta_b} L^i &= \lambda_k \nabla_{\theta_b} L_k^i + \lambda_s \nabla_{\theta_b} L_s^i
\end{aligned}
$$

The weights for the next iteration θ_b^{i+1}, θ_s^{i+1}, and θ_k^{i+1} are obtained with an optimizer step such as Adam (Kingma and Ba, 2015).

[7] D_s and D_k can be the same dataset, i.e., we have experiments where we use Librispeech for both. This still implies that two batches are sampled from the dataset independently during each optimization step.

Joint training

Most work on MTL assumes training can be done with a "joint" forward step, namely each sample has labels for all tasks. As a baseline, we want to see if training with joint forward steps is effective for MTL of ASR and SKR. We tried two settings of joint optimization. The first setting uses only data from Librispeech, which has both speech and speaker labels. We do not expect this to be effective due to the aforementioned lack of session variability in Librispeech. The second setting uses an ASR model to generate pseudo-labels for the VoxCeleb2 dataset. We decided to do this with the `BASE`[8] Whisper (Radford et al., 2023) model. We remove any data labeled as non-English by Whisper, and normalize the transcripts to the character vocabulary of Librispeech.

4.2.3 Length of audio input during training

We note that the discrepancy between audio input lengths for training ASR and SKR systems is a potential issue, as the network will observe drastically different sequence lengths for each task. We therefore suggest two strategies for cropping the speaker recognition audio segments. The first strategy follows the current paradigm (Snyder et al., 2018; Chung et al., 2020; Desplanques et al., 2020; Lin and Mak, 2020; Vaessen and van Leeuwen, 2022) and uses short crops of 2 s. The second strategy is to use crops of 10 s, a value close to the average length of audio in Librispeech.

4.3 Experiments

4.3.1 Data

We used the Librispeech (Panayotov et al., 2015) dataset to train and evaluate for speech recognition. The dataset consists of utterances from audio books, read by volunteers. We used all three train subsets, for a total of 960 hours of training data with 2484 speakers, indicated as `ls`. The training audio utterances have a mean of 12.3 seconds, and a standard deviation of 3.84 seconds. To minimize right-padding (with 0) in the speech batches, a batch was collected by sampling utterances with similar length. We used the `dev-other` subset to determine a validation word error rate (WER_{val}). Evaluation was done on the difficult `test-other` subset. The transcriptions were greedily decoded, we did not use a language model. We also create a trial list for `dev-other` and `test-other` in order to evaluate speaker recognition performance on Librispeech. We use all possible speaker pairs in the respective subset, but exclude positive trials from the same session (book), and only include same-sex negative trials.

The VoxCeleb1 (Nagrani et al., 2017) and VoxCeleb2 (Chung et al., 2018) datasets were used to train and evaluate on speaker recognition. The datasets consist of videos of celebrities taken from YouTube. Each speaker has multiple recordings (videos), and

[8] with https://pypi.org/project/openai-whisper/

each recording has multiple utterances. The VoxCeleb2 development set was used as to create a training, validation, and development split. It has a total of 2305 hours of data, with 5994 speakers, and a mean utterance length of 7.79 seconds and a standard deviation of 5.22 seconds. We held-out 194 speakers (97 male/female) to create a development subset `vox2-dev`. The remaining 5800 speakers were used as training data, indicated as `vc`. To be able to compute a validation EER (EER_{val}) on `vox2-dev` we randomly sampled 100 k positive and 100 k negative trial pairs, while ensuring that negative trials are same-sex, and positive trials are from two different sessions. The final evaluation of experiment was done on the VoxCeleb1 dataset. We used the "original" and "hard" trial list, dubbed respectively `vox1-o` and `vox1-h`. `vox1-o` has only 40 speakers, and is the original test set of VoxCeleb1. On the other hand, `vox1-h` has 1190 speakers, includes all the data from the train *and* test set of VoxCeleb1, and each negative trial pair has the same sex and nationality. There is no speaker overlap between VoxCeleb1 and VoxCeleb2. During training and validation, all utterances are randomly cropped to either 2 or 10 seconds. During evaluation, we use the full length of the utterance unless stated otherwise. Trials are scored by computing the cosine similarity between two speaker embeddings, without any further processing.

To test on out-of-distribution (OOD) data, we also evaluate speech recognition on the English part of `HUB5` 2000,[9] and speaker recognition on NIST SRE08 (Martin and Greenberg, 2009). For `HUB5`, we segment the audio based on the ground truth reference to make evaluation easier. We also pre-process the text by removing all annotations and normalizing to the Librispeech character vocabulary. For `SRE08` we only make use of the first condition, where each trial has an enrollment and test segment with a length of 10 seconds. For both datasets we resample the audio to 16 kHz.

4.3.2 Training protocol

We use the following training protocol, unless stated otherwise, to balance between spending an equal amount of computational resources on each method, and limiting the required computational budget. Each network variant under study is initialized with available[10] self-supervised, pre-trained weights (Baevski et al., 2020a), with an identical random seed for all experiments. We use a batch size of up to 3.2 M audio samples (≤ 200 seconds) for both tasks (Baevski et al., 2020a). We use the default regularization methods for wav2vec 2.0, namely LayerDrop (Huang et al., 2016; Fan et al., 2020), Dropout (Srivastava et al., 2014), and SpecAugment masking (Park et al., 2019). The optimizer is Adam (Kingma and Ba, 2015) using learning rate η which is varied according to a tri-stage learning rate schedule (Baevski et al., 2020a) with 200 k steps. For a given learning rate $\eta_{\max}$, this schedule consists of a warm up phase of 20 k steps where η is linearly increased from $\frac{1}{100}\eta_{\max}$ to $\eta_{\max}$, followed by a constant phase of 80 k steps where $\eta = \eta_{\max}$, and a decay phase of 100 k steps where cosine annealing is used to decrease from $\eta_{\max}$ to $\frac{1}{20}\eta_{\max}$. We clip gradients

[9] Available at https://catalog.ldc.upenn.edu/LDC2002T43

[10] The pre-trained weights were retrieved from https://huggingface.co/facebook/wav2vec2-base.

to the range $[-1, 1]$. In the disjoint MTL setting clipping is done before summing the gradients for the respective tasks unless stated otherwise. For the first 3 k steps the whole wav2vec 2.0 network is frozen, only the heads are updated (Baevski et al., 2020a). The feature extractor CNN is always frozen (Baevski et al., 2020a). We use CTC loss (Graves et al., 2006) as the speech recognition loss, and AAM softmax loss (Deng et al., 2019; Liu et al., 2019) for the speaker recognition loss with a scale of 30 and a margin of 0.2 (Desplanques et al., 2020). For each network variant we perform a grid search over the learning rates $\{1, 3\} \times 10^{-\{4,5,6\}}$, We validate every 5 k steps, and stop early if the validation loss has not decreased for 8 times in a row i.e., 40 k steps. For the evaluation, we select a learning rate based on $\frac{1}{4}\text{WER}_{\text{val}} + \frac{3}{4}\text{EER}_{\text{val}}$ to roughly equate the ranges of the WER and EER values, which leads to a fair balance between both tasks in the hyperparameter search. Training is done on a machine with 12 CPU cores, 32 GB of RAM, and a single GPU[11] using at most 24 GB of VRAM. In total 313 days of GPU time was spent on experiments.

4.3.3 Comparing MTL optimization strategies

The first set of experiments are focused on comparing optimization strategies and are shown in Table 8. The network architecture in these experiments is fixed; the speech and speaker head both use $\mathcal{C}^{12}$, and the speaker head uses `mean` pooling. For each MTL strategy we train with either Librispeech or a combination of Librispeech and VoxCeleb2. We also vary the loss weights λ_s and λ_k to be equal (each 0.5) or favoring speech (respectively 0.9 and 0.1). For disjoint MTL training, we also vary whether the SKR batch has 2 s samples or 10 s samples. We evaluate each task on an in-domain and out-of-domain test set. Note that for speaker recognition evaluation we compute the speaker embedding over the full test utterance.

First, we observe that single-task training for SKR with Librispeech compared to VoxCeleb2 achieves much worse performance on the in-domain evaluation set `vox1-h` and the out-of-domain evaluation set `SRE08`. This pattern repeats, where for joint and disjoint MTL we also observe worse SKR performance when only Librispeech is used. When we do joint optimization with whisper-transcribed VoxCeleb2, the speaker recognition performance drastically improves. Note that MTL training using only Librispeech data has worse performance than STL training with only LS data, for both SKR and ASR. Looking at disjoint MTL training, we see that using 2 s SKR chunks during training seemingly leads to no speaker recognition capabilities (discussed further in Section 4.3.5). Using 10 s SKR chunks, however, makes the MTL outperform the STL baseline on the `vox1-h` test set, but also a slightly degraded ASR performance on `test-other`. We also see that the choice of $\lambda_s = 0.9$ versus $\lambda_s = 0.5$ is a trade-off between SKR and ASR performance. Lastly, we observe that all MTL models have drastically degraded performance on out-of-distribution evaluation data (`HUB5`, `SRE08`) compared to the STL baselines.

[11] Experiments were done with NVIDIA A5000, A6000 and A100 GPUs.

Table 8: *Comparison of STL baselines versus joint and disjoint MTL training. Evaluation is done on in-distribution (`test-other`, `vox1-h`) and out-of-distribution (`HUB5`, `SRE08`) datasets. The second column indicates which training data was used. * indicates ASR labels generated with Whisper.*

		ASR (WER %)		SKR (EER %)	
network	train data	test-other	HUB5	vox1-h	SRE08
STL (baseline)					
ASR	ls	10.4	40	-	-
ASR	vc*	16.6	25	-	-
SKR (2s)	ls	-	-	33	42
SKR (2s)	vc	-	-	5.1	16
MTL (joint, full length samples)					
$\lambda_s = 0.5$	ls	15.3	48	36	40
$\lambda_s = 0.5$	ls+vc*	18.1	36	10.3	24
$\lambda_s = 0.9$	ls+vc*	17.5	36	7.2	26
MTL (disjoint, 2 sec SKR samples)					
$\lambda_s = 0.5$	ls	14.5	54	45	46
$\lambda_s = 0.5$	ls+vc	11.1	46	41	45
$\lambda_s = 0.9$	ls+vc	11.5	48	42	46
MTL (disjoint, 10 sec SKR samples)					
$\lambda_s = 0.5$	ls	13.6	49	36	44
$\lambda_s = 0.5$	ls+vc	11.1	80	4.8	39
$\lambda_s = 0.9$	ls+vc	11.2	84	4.7	27

4.3.4 Varying architectures

In the second set of experiments we focus on different strategies for extracting speaker embeddings for the SKR task, and effectively combining it with the speech information for ASR. For all MTL experiments we use $\lambda_s = 0.5$ and only train with disjoint steps. We train with either 2 s or 10 s SKR chunks, and for MTL training we place the speaker head at either $\mathcal{C}^6$ or $\mathcal{C}^{12}$. The speech head is always at $\mathcal{C}^{12}$. We also applied gradient clipping after summing the gradients instead of before.[12] The results can be seen in Table 9. The first observation, unexpected based on contemporary work in SKR, is that in the STL setting the speaker recognition performance was actually

[12]This was a configuration error. Clipping after summation changes how the λ values in the loss function affect optimization. We did not repeat the experiments with the correct configuration because we believe that this will not change the results in a major way, with the added benefit of preventing the release of some CO_2 into the atmosphere.

Table 9: *Comparing three methods to extract speaker embeddings from wav2vec 2.0. Evaluation is done on in-distribution (`ls-to`, `vox1-h`) and out-of-distribution (`HUB5`, `SRE08`) datasets. We vary training with 2 s or 10 s chunks, and for MTL also using $\mathcal{C}^6$ or $\mathcal{C}^{12}$ as input to the speaker recognition head. We used `ls+vc` as MTL training data.*

	ASR (WER %)		SKR (EER %)	
SKR head	`test-other`	`HUB5`	`vox1-h`	`SRE08`
STL, `x/x` implies 2 s/10 s SKR samples				
`mean`	-	-	5.1/5.1	17/13
`first`	-	-	5.4/5.2	19/14
`ecapa`	-	-	6.3/5.8	21/13
MTL disjoint, 2 sec SKR samples, `x/x` implies $\mathcal{C}^6/\mathcal{C}^{12}$				
`mean`	13.5/13.4	53/52	21/34	40/44
`first`	13.6/13.9	52/53	12/34	29/40
`ecapa`	13.2/13.9	45/53	9/35	25/39
MTL disjoint, 10 sec SKR samples, `x/x` implies $\mathcal{C}^6/\mathcal{C}^{12}$				
`mean`	12.9/12.8	51/79	3.9/4.0	31/33
`first`	13.2/13.4	46/79	3.9/4.0	15/16
`ecapa`	13.2/12.7	42/83	4.2/4.7	19/16

better when using 10 s chunks during training. This is more noticeable on the `SRE08` data. Secondly, the specific variant of the speaker head has only a minor effect on the ASR performance. However, using `ecapa` alongside $\mathcal{C}^6$ seems very effective compared to `mean` or `first` pooling. Noticeably, when training with 2 s second chunks, using $\mathcal{C}^6$ instead of $\mathcal{C}^{12}$ seems to result in some SKR capabilities. Lastly, ASR performance on `test-other` is worse compared to Table 8 with equivalent settings, likely due to the change in the gradient clipping strategy.

4.3.5 Different evaluation conditions

In the last set of experiments we further analyze the results described in Table 8. As we observed decreased performance on out-of-distribution data for the MTL models, we also wanted to observe the performance on cross-disjoint-task data, namely, can we do SKR on Librispeech data, and ASR on VoxCeleb2 data? To evaluate for ASR on VoxCeleb2 we use the smaller `vox1-o` set and use the pseudo-labels from Whisper as the reference. In Table 10 we see that cross-disjoint task performance is lacking. Noticeably, disjoint MTL with 10 s chunks has a 100 % WER on `vox1-o` and a 42 % EER on `test-other`. Furthermore, we observed that MTL disjoint training with 2 s SKR chunks and `mean` pooling did not show any SKR capabilities. Therefore, perhaps counter-intuitively, we also evaluate on SKR by only using the *first* 2 s of the utterance,

Table 10: *Evaluation of STL and MTL models on cross-disjoint-task and out-of-distribution data. MTL models are trained with* $\lambda_s = 0.9$ *and* `mean` *pooling. The joint model is trained* `ls+vc*`*, * indicates ASR labels from Whisper. The disjoint models are trained with* `ls+vc`*. For ASR evaluation on Voxceleb we use the* `vox1-o` *test set, with labels from Whisper. For SKR evaluation we show results using only the first 2 seconds, or the full utterance of each audio file.*

	STL		MTL		
evaluation	ASR	SKR	joint	disjoint (2 s)	disjoint (10 s)
ASR (WER in %)					
`test-other`	10.4	-	17.5	11.5	11.2
`vox1-o`	35	-	27	35	100
`HUB5`	40	-	36	48	84
SKR, full sample evaluation (EER in %)					
`test-other`	-	2.2	8.5	40	42
`vox1-h`	-	5.1	7.2	42	4.7
`SRE08`	-	16	26	46	27
SKR, 2 s sample evaluation (EER in %)					
`test-other`	-	4.9	13.4	7.9	44
`vox1-h`	-	11	21	12.4	16
`SRE08`	-	32	41	33	41

instead of the whole utterance. With this evaluation strategy, we observe that for STL SKR, MTL joint, and MTL disjoint with 10 s chunks, the SKR performance is worse compared to using the whole audio file. However, MTL disjoint training with 2 s chunks has decent performance when also evaluating with 2 s of audio. This compares to no capabilities when evaluating on the full sample.

We also analyzed how the speaker information differs throughout the transformer layers of wav2vec 2.0. This was done by first training a model in a specific setting, and then evaluating the output of *each* layer independently, with `mean` pooling as the method for speaking embedding extraction. The evaluation was done on the `vox2-dev` set and trial list. We show the results in Figure 3. First, we see that self-supervised weights contain little speaker information, although there is a slight increase until the last two layers, which seem to remove most speaker information. In the single-task learning setting, we see a gradual improvement each layer, although there is some plateauing around layer 7, 8 and 9. The disjoint MTL model with 10 s chunks and using $\mathcal{C}^{12}$ seems to follow the STL line closely. Moreover, we see that MTL models with a speaker head at $\mathcal{C}^{6}$ actually lose speaker information after the 6th layer, indicating the models attempt to separate speech and speaker information. Lastly, the joint

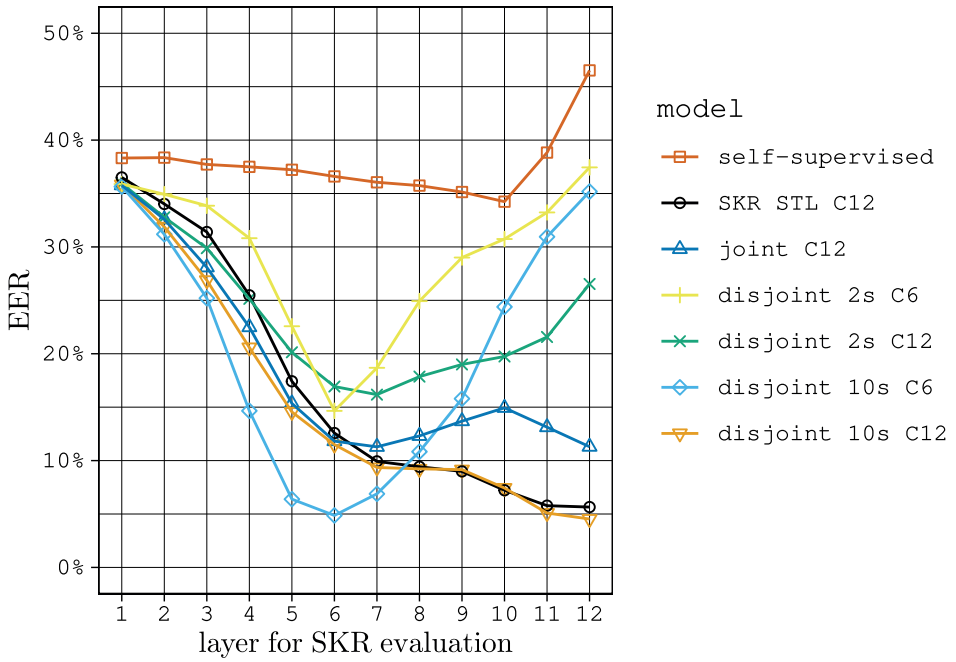


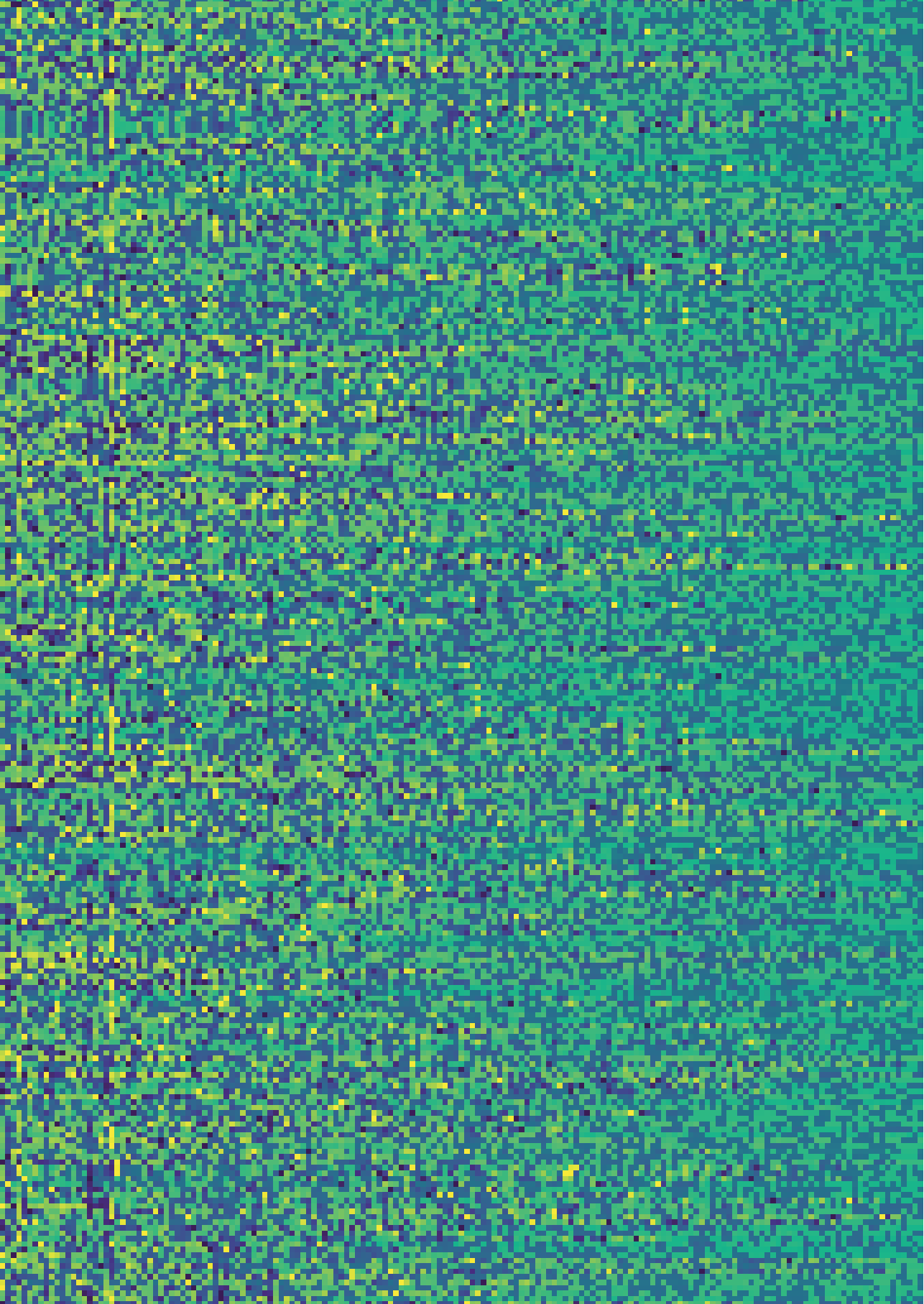
Figure 3: *The performance of each wav2vec 2.0 transformer layer by* `mean` *pooling the output sequence before (i.e., only self-supervised weights) and after fine-tuning wav2vec 2.0 (STL and 5 MTL variants). MTL models use either* $\mathcal{C}^6$ *or* $\mathcal{C}^{12}$ *as input to the speaker recognition head* during training*, but change the layer during evaluation, i.e., we vary* $\mathcal{C}^n$ *with* $n \in [1, 12]$ *at test time. Evaluation is done on* `vox2-dev`*.*

MTL model has interesting behavior, where performance starts decreasing at layer 7, but starts improving again at layer 11 and 12.

4.4 Conclusion

We have shown that creating a MTL model for speech and speaker recognition is challenging. First, we need multi-labeled data with session variability, as Librispeech is insufficient for creating a good SKR model. Our mitigation strategies with either pseudo-labels, or disjoint training, have degraded ASR performance on in-domain test data. Moreover, optimizing a model with disjoint steps does not generalize to out-of-distribution data. We further saw that MTL models have increased SKR performance, at the cost of decreased ASR performance. It is difficult to include speaker information without harming ASR performance. In our MTL setup we did not see that speaker information benefitted ASR performance. This is different from prior work (BenZeghiba and Bourlard, 2003; Peddinti et al., 2015), as we wanted the model to implicitly generate the speaker information throughout the feed-forward process, instead of explicitly adding it as input to the network. This might be inherent

to the MTL loss function, which always needs to trade-off the CTC loss versus the AAM-softmax loss. We believe that future work could focus on integrating speaker information into the CTC loss, by adding e.g., speaker-related targets, and foregoing the need to use two loss functions and two output heads. We did see promising results with the `ecapa` strategy alongside using $\mathcal{C}^6$ as input. This indicates that there is only a limited amount of weights which should be shared between the tasks. Thus, future work could perhaps also analyze this kind of shared architecture more deeply, where the focus is on a smaller shared backbone network, and larger task-specific heads. Most importantly, we think that research into MTL of ASR and SKR can benefit significantly from future work creating a proper dataset, with high speaker session variability and accurate speech labels.



5 The effect of batch size on contrastive self-supervised speech representation learning

In which our adventurer uncovers the relationship between the amount of data observed during self-supervised pre-training, and the subsequent performance when fine-tuning on downstream tasks.[13]

Foundation models have become the norm in deep learning research. In the audio domain, popular models with open weights include wav2vec 2.0 (Baevski et al., 2020a; Conneau et al., 2021), HuBERT (Hsu et al., 2021), and WavLM (Chen et al., 2022a). These transformer models all use a form of self-supervised learning (SSL) with the use of a pretext task to learn ("pre-train") speech representations. The models can then be fine-tuned on a myriad of downstream tasks (Yang et al., 2021), including speech recognition, speaker recognition, emotion recognition, and intent classification. However, self-supervised pre-training takes a tremendous amount of resources, exceeding high-end consumer grade hardware at the time of writing. First, due to the unlabeled nature of self-supervision, it is relatively cheap to increase the dataset size. Over a span of two years, we have seen public training datasets increase by two orders of magnitude,[14] with wav2vec 2.0 using 1 k hours of audio (circa 100 GB) from Librispeech (Panayotov et al., 2015), to WavLM using 94 k hours (circa 10 TB) by combining Libri-light (Kahn et al., 2020), GigaSpeech (Chen et al., 2021) and VoxPopuli (Wang et al., 2021). Second, the seminal works mentioned above all report results of models trained with large batch sizes using data parallelism across many GPUs. For example, for models with 94 M parameters, WavLM and HuBERT use 32 GPUs, and wav2vec 2.0 uses 64 GPUs, with batch sizes of respectively 3 hours, 45 minutes, and 90 minutes of audio. The number of GPUs needed to work with these

[13]Research work based on this chapter is currently submitted for double-blind peer review.

[14]Excluding Whisper (Radford et al., 2023) and Google USM (Zhang et al., 2023), with respectively 680 k hours and 12 M hours of private training datasets.

large batches in a timely manner, as well as the required disk space for the datasets, make it non-trivial to apply these algorithms.

While the effect of dataset size on performance is (at least partially) known (Baevski et al., 2020a; Conneau et al., 2021), to our knowledge there are no studies on the scaling behavior of SSL algorithms with respect to the batch size and number of training iterations. This can be of interest to researchers who do not have the resources to study these algorithms under large batch size conditions, or practitioners who need to make a trade-off between time, computational budget, and desired performance. Given a fixed model complexity, dataset size, and number of training iterations, how much is gained by increasing the batch size? How well do these techniques work with fewer resources, and can the academic community do meaningful experiments without industrial-scale data centers? Although we aim to answer these questions generally, for precisely the reason of available computational resources, we limit ourselves to studying the wav2vec 2.0 (Baevski et al., 2020a) model extensively, leaving other SSL methods for future work. Concretely, we set out to address the following research questions:

RQ 1. How does the batch size affect the pre-training procedure of wav2vec 2.0?
RQ 2. How does the batch size during pre-training of wav2vec 2.0 affect downstream fine-tuning?
RQ 3. Can we compensate for a reduction of the batch size by increasing the amount of training iterations by the same factor?

Regarding all three RQs, and given the existing literature on speech SSL (Baevski et al., 2020a; Hsu et al., 2021; Chen et al., 2022a), we hypothesize that large batch sizes are essential for pre-training convergence and the model's ability to be properly fine-tuned to the downstream task. For RQ 1, we are interested in knowing whether a large batch size is a necessity for optimizing the objective. It would be valuable to know the smallest possible converging batch size, and how optimization behaves with this batch size compared to the canonical, large batch size. For RQ 2, we expect that a larger batch size will lead to better downstream task performance, but we are especially interested in how much the performance improves with each doubling of the batch size. What is the minimum batch size at which we see that fine-tuning is possible, and how does this depend on the amount of labeled data available for fine-tuning? For RQ 3, we are interested in knowing whether training twice as long with half the batch size results in the same performance. There is evidence that contrastive methods benefit from large batch sizes (Chen et al., 2022b), although in wav2vec 2.0 the quantity of potential negatives samples does not increase with the batch size, only with the sequence length. Moreover, according to work on optimizer scaling laws (Goyal et al., 2018; Malladi et al., 2022), with a fixed number of epochs, learning rates can be adjusted accordingly with the batch size to obtain a very similar optimization trajectory. Therefore, we hypothesize that performance for wav2vec 2.0 is only a function of how much data is seen during self-supervision, and that with patience, people with fewer resources can also carry out pre-training.

To answer these questions, we pre-train wav2vec 2.0 with batch sizes ranging from 87.5 seconds to 80 minutes. We then fully fine-tune these models for speech recognition (updating most weights), with 10 minutes to 960 hours of labeled speech. To include other speech technology tasks, we also fine-tune following the SUPERB benchmark protocol, where small downstream models are trained on different categories of speech tasks. Here, the foundation model weights are frozen, and the (trainable) weighted sum of all layer outputs of the foundation model are used as input. Hereby, we make the following contributions:

1. We perform a comprehensive study of the effect of batch size and amount of training iterations for pre-training wav2vec 2.0, helping practitioners to make trade-offs when deciding on downstream task performance.
2. We show that the most important factor for the downstream task performance is the amount of data seen during self-supervision, indicating that fixing the product of batch size and training iterations in a benchmark can provide valuable information.
3. We provide the pre-training model checkpoints, with an interval of 5 k steps, for further analysis.[15]

The rest of this chapter is structured as follows. First, we will cover related work in Section 5.1, including studies on batch sizes with stochastic gradient descent, (contrastive) SSL (in speech) and its scaling behavior, and research on SSL with smaller budgets. Then, Section 5.2 will explain wav2vec 2.0 pre-training and fine-tuning, followed by experimental setup and results in Section 5.3, and we will close with a discussion and conclusion in Section 5.4 and Section 5.5.

5.1 Related work

5.1.1 Stochastic gradient descent and large batch sizes

McCandlish et al. (2018) studied the trade-off between time and computational resources when choosing a batch size. It is argued that a small batch size leads to gradients dominated by noise, which is averaged out by consecutive update steps, or more efficiently, by using data parallelism. However, when a batch size is very large, the gradient estimate contains little noise, and therefore sampling two (large) batches and averaging their gradient will not lead to a significantly better estimate. In this case doubling the batch size does not serve a practical purpose anymore. Thus, there is a critical batch size, the exact value varying for each task and domain, after which an increase in batch size has strongly diminishing returns. Complementarily, Shallue et al. (2019) conducted a study on the batch size affecting generalization performance, across multiple datasets (5 vision, 2 text), neural network families (FC, 3 CNNs, LSTM, Transformer) and optimizers (SGD, with (Nesterov) momentum). They experimentally confirm the existence of a critical batch size, and observe the

[15]See the code repository in Research Data Management for a link to the model checkpoints.

magnitude of this critical batch size depends on the dataset, neural network type, and optimizer, but no clear relationship is found. With respect to the optimization trajectory, Smith et al. (2018) showed that a decaying learning rate schedule is equivalent to increasing the batch size during training, up to a batch size around 10 % of the training dataset size. There is also work on a *linear* scaling law for SGD by Goyal et al. (2018) and *square root* scaling law for adaptive methods such as Adam by Malladi et al. (2022). These laws suggest that, for a fixed number of epochs, the same optimization trajectory can be obtained with different batch sizes by adjusting the learning rate.

5.1.2 Self-supervised speech representation learning

Representation learning, as defined by Goodfellow et al. (2016), concerns itself with being able to encode information in such a way that it makes learning a subsequent downstream task straightforward. For speech, a good representation would allow, e.g., phonemes to be linearly separable in order to perform speech recognition, or speaker attributes to be distinctly clustered, so that a distance metric can be used to recognize speakers. Models which are trained in a supervised fashion already learn task-specific representations, as usually the output of a penultimate layer is used for classification purposes. In self-supervised representation learning, a pretext task is used instead, with the hope that solving this task requires learning representations which are also helpful for learning the actual downstream task(s) of interest. These pretext tasks are designed such that they use some property of the input data itself as a label. This allows using much larger, and cheaper to collect, datasets, and for representations to potentially be useful for multiple distinct processing tasks. A good overview of pretext tasks used for speech representation learning is given by Mohamed et al. (2022). They define three categories of pretext tasks, namely *reconstructive*, *contrastive*, and *predictive*.

Reconstructive pretext tasks limit the view of the speech signal, whereafter the model needs to fill in or complete the signal in some fashion. Models using a reconstructive pretext task include VQ-VAE by van den Oord et al. (2017), Mockingjay by Liu et al. (2020), the DeCoAR variants introduced by Ling et al. (2020) and Ling and Liu (2020), and TERA by Liu et al. (2021). Later, Mohamed et al. (2022) argue that reconstruction of speech results in heavily entangled representations, which makes them less useful for downstream tasks.

Contrastive pretext tasks also limit the view of the speech signal. However, instead of simply reconstructing, the objective focuses on predicting what information should or should not be encoded at a certain (unseen) time step of the signal. This is done with an anchor representation, for which a target representation (which should be there) is distinguished from distractor representations (which should *not* be there), thus creating contrast between learned representations. Wav2vec 2.0 is a popular contrastive model, which this thesis focuses on exploring more deeply. Other contrastive models include Unspeech by Milde and Biemann (2018), CPC by van den Oord et al. (2018),

and other wav2vec variants, as presented by Schneider et al. (2019), Baevski et al. (2020b), Sadhu et al. (2021) and Conneau et al. (2021). The challenge of contrastive models, according to Mohamed et al. (2022), is the sampling of distractors. For example, the learned representations can become invariant to speaker information if they are sampled from the same utterance. Moreover, it is not clear how relatable the target and distractors are to the anchor, due to the difficulty of segmenting speech signals.

Finally, predictive pretext tasks have known targets for the parts of the speech signal which were hidden. It is these targets that the models learn to predict. For example, HuBERT by Hsu et al. (2021) uses cluster centroids of MFCCs computed over the training dataset as targets during self-supervision. At one or more points during training, new targets are generated, by clustering the hidden representation outputs from the model, replacing the initial MFCC-based clusters. Other examples of predictive approaches are WavLM by Chen et al. (2022a), which is similar to HuBERT, but adds background speech to the input data which the model needs to learn to ignore, and data2vec by Baevski et al. (2022) as well as DinoSR by Liu et al. (2023), which use a teacher-student approach, where targets are provided by a teacher model, which is updated with an exponentially moving average of the student model weights. As argued in Mohamed et al. (2022), the challenges of the mentioned models are mostly computational; HuBERT and WavLM require multiple iterations and good initial targets, whereas teacher-student approaches require twice as many model parameters during training.

5.1.3 Scaling self-supervised representation learning

Kaplan et al. (2020) analyse the scaling behavior of pre-training large language models, with respect to model size, dataset size, and the number of training steps. Their primary finding is that the test loss follows a power law in relation to all three aspects, as long as model size is increased according to the dataset size, and training length is not made a bottleneck. It is also found that very large models are more sample efficient, i.e., fewer iteration or less data is required compared to smaller models. Goyal et al. (2019) study the scaling behavior of visual representation learning, with respect to model size, dataset size, and complexity of the pretext task. They found that increasing both the dataset size, and complexity of the pretext task, is beneficial, as long as the model size is large enough. For speech SSL, the scaling behavior of the model size and the fine-tuning dataset size is studied by Pu et al. (2021). They use the reconstructive pretext task Mockingjay by Liu et al. (2020), and their results match Kaplan et al. (2020); larger model size leads to better performance, and larger models require less fine-tuning data.

5.1.4 Contrastive learning and batch size

Contrastive self-supervision benefits from large batch sizes, as ablated by Chen et al. (2020) who introduced the SimCLR method, and shown by, e.g., the method CLIP

introduced by Radford et al. (2021) and Florence method introduced by Yuan et al. (2021b). A hypothesis for this observation is that distractors are often sampled within the same mini-batch, and thus more (and potentially better) distractors are available as the batch size increases. However, Mitrovic et al. (2020) show that computing the contrastive objective with fewer (e.g., only two) distractors per anchor leads to better performance, indicating that large batch sizes are the key factor of improved performance, and not the amount of available negative samples. Chen et al. (2022b) argue that small batch sizes in contrastive learning suffer from a gradient bias, which large batches sizes alleviate. Note that in wav2vec 2.0, negative samples are only taken from the same utterance. The batch size does not have any effect on the quality and quantity of negative samples, so there might be a gradient bias even with large batch sizes.

5.1.5 Self-supervised learning with academic budget

The apparent effectiveness of large batch sizes makes it difficult to do research without a large computational budget. There has been some work on trying to reduce the resources required to do pre-training. For example, Izsak et al. (2021) pre-trained a BERT model to nearly equivalent performance with only 8 GPUs (with 12 GB VRAM) in 1 day, compared to 4 days with 16 TPUs (having 32 GB RAM) in the original work by Devlin et al. (2019). This was done by reducing the maximum sequence length, focusing on large models, pre-masking data, and using specialized software packages such as DeepSpeed by Rajbhandari et al. (2020) and Apex by Micikevicius et al. (2018). Similar work has been done for the HuBERT model by Chen et al. (2023). They show that using target representations from a fine-tuned ASR model in the first iteration of HuBERT pre-training (instead of MFCCs) leads to better performance, while needing fewer GPU hours. Another line of thinking is presented by Cao and Wu (2021), where it is shown that self-supervised learning in vision can be done on small datasets, with low resolution images, and with models with relatively few parameters.

5.2 Methodology

We will describe the architecture and pre-training procedure of wav2vec 2.0 (Baevski et al., 2020a), explicitly mentioning details we found to be essential for performance that received less attention in the original paper. We provide a schematic overview of wav2vec 2.0 in Figure 4, which can aid in understanding the dependence of different components and mathematical variables of the framework, which we will introduce below.

5.2.1 The CNN + Transformer network for audio

In this chapter we use the standard architectural setup for self-supervised learning with audio (Baevski et al., 2020a; Hsu et al., 2021; Chen et al., 2022a). First, the

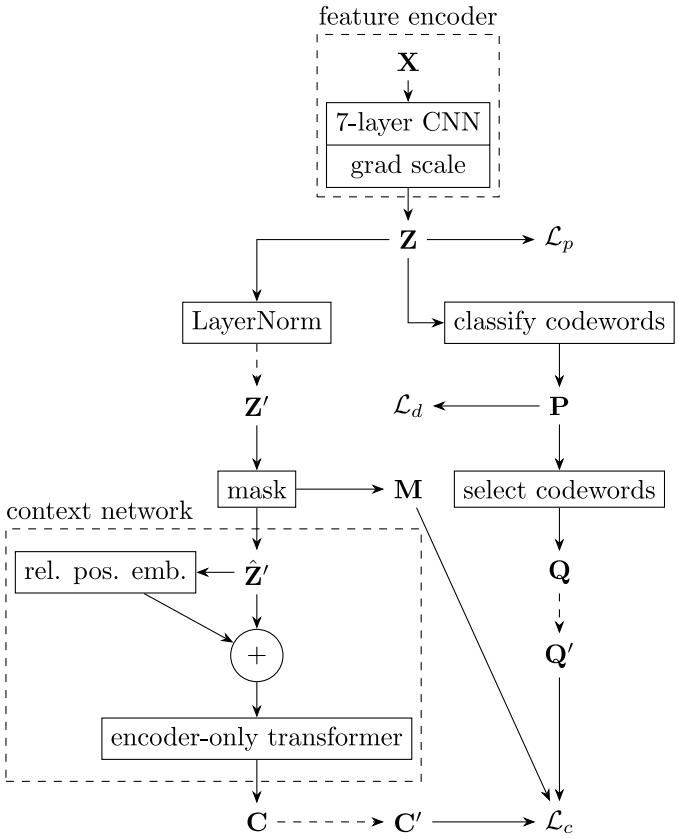


Figure 4: *Schematic overview of the wav2vec 2.0 framework during self-supervision. Dashed arrows indicate a projection using a linear layer without activation to match a target dimension.*

raw audio is processed into speech features with a 1-d convolutional neural network (CNN) called the *feature encoder*. This network has 7 convolutional layers, all of them with 512 channels. For each respective layer, the kernel sizes are [10, 3, 3, 3, 3, 2, 2] with strides [5, 2, 2, 2, 2, 2, 2]. We include padding of [3, 1, 1, 1, 1, 0, 0] on both sides in order to change the output rate of the CNN to 50 feature vectors per second of audio, instead of slightly less as in (Baevski et al., 2020a). Each layer is followed by GELU activation, with tanh approximation. For the first convolutional layer, each output channel (thus, along the time dimension) is independently normalized to a learned mean and variance, by using GroupNorm (Wu and He, 2018) with 512 groups and a learnable affine transform, before applying the activation function. The CNN is followed by a gradient scaling layer, which during the forward pass acts as an identity function, but during the backward pass multiplies the global gradient with a constant. Hereby the magnitude of the gradients of the CNN can be controlled. The gradient scale constant is set to $\frac{1}{10}$ (Baevski et al., 2020a). Mathematically, the feature encoder

$f(\cdot)$ processes the raw 16 kHz audio segment $\mathbf{X} = \mathbf{x}_1, \ldots, \mathbf{x}_r$ to a sequence of latent speech feature vectors $\mathbf{Z} = \mathbf{z}_1, \ldots, \mathbf{z}_T$, with $T = \lfloor \frac{r}{320} \rfloor$.

The latent speech feature vectors are a local representation of the speech signal, in that each vector encapsulates speech factors in a window of 20 ms. A vanilla encoder-only transformer network called the *context network* is used to create contextualized representations based on the local representations. Due to self-attention in the transformer network, a contextual representation encapsulates speech factors from the entire audio segment. In the `BASE` configuration, the transformer network has 12 layers, with an hidden dimension of 768 in the self-attention module, 12 attention heads, and a scale-up to 2048 dimensions in the feed-forward network, with GELU activation. In the `LARGE` configuration, there are 24 layers, 1024 hidden dimensions, 16 attention heads, and a scale-up to 4096 dimensions. LayerNorm (Lei Ba et al., 2016) is applied after the residual self-attention and feed-forward operations, such that the contextual representations follow a multivariate normal distribution with a learned mean and variance, before a learned transformation is applied.

The local representation sequence $\mathbf{Z}$ cannot directly be used as input to the transformer network. First, LayerNorm is applied, so that each feature vector $\mathbf{z_i} \in \mathbf{Z}$ follows a learned multivariate normal distribution. Then, a single linear layer (without activation) projects the local representation from 512 to 768 dimensions. The projected representations are then masked, which is described in more detail in Section 5.2.2. The masking is an important aspect for pre-training, but is also beneficial during fine-tuning. During inference no masking is done. As transformers need explicit positional information, a relative positional embedding is computed from the masked latent vectors, with a single convolutional layer, followed by GELU activation. The convolutional layer has 768 output channels, a kernel size of 128, padding of 64 on both sides, and 16 groups. Moreover, weight normalization (Salimans and Kingma, 2016) is applied on the kernel weights to aid convergence speed. The stride is 1, therefore a relative positional embedding can be added to each latent vector. Due to this summation, the input vectors of the transformer have a context of 1.25 seconds from both sides.

To regularize the network, dropout is applied on 3 locations[16] in the transformer layers, namely on the self-attention scores (before weighted sum of values), on the output of the self-attention module (before residual addition), and on the output of the feed-forward network (before residual addition).

To allow for independent modifications to the dimensions of each component in wav2vec 2.0, there are three so-called projection layers, consisting of a single fully-connected layer without activation. In our notation we will use the accent $'$ to indicate a projected vector. The first projection layer is before the context network

[16]In (Baevski et al., 2020a) dropout is also applied on the latent speech features, after the projection layer. Additionally during SSL only, it is applied before quantization as well. They also apply LayerDrop, which we skip to simplify the data-parallel implementation.

$g(\cdot)$, where the projection operates on *normalized* $\mathbf{z}$. This $\mathbf{z}'$ is masked by a masking function $m(\mathbf{z}')$. Then, the context network processes the masked local representations $\hat{\mathbf{Z}}' = \hat{\mathbf{z}}'_1, \ldots, \hat{\mathbf{z}}'_T$ to contextual representations $\mathbf{C} = \mathbf{c}_1, \ldots, \mathbf{c}_T$. We will write $\mathbf{C} = g\left(\hat{\mathbf{Z}}'\right)$ to indicate that each representation $\mathbf{c}_t$ had access to the entire sequence $\hat{\mathbf{Z}}'$. The other locations where projection occurs is in the the computation of the contrastive loss, which we will describe below.

5.2.2 Self-supervision with contrastive learning

In this chapter we study only one self-supervised learning approach for speech representation learning, namely wav2vec 2.0 (Baevski et al., 2020a). Intuitively, the pretext task is to mask multiple regions of $\mathbf{Z}$, the local speech representation sequence, and feed this into the context network to predict the masked-out representations. By enforcing contrast, i.e., dissimilarity, between predicted representations, the feature encoder has to place different information at different locations in the sequence $\mathbf{Z}$. This can be seen as an intrinsic bias to learn phonetic units, as phones are also expected to differ throughout an utterance of speech. However, the context network does not directly predict masked $\mathbf{z}_t$ values. Instead, each $\mathbf{z}_t$ is mapped to a quantized vector $\mathbf{q}_t$. This $\mathbf{q}_t$ is part of a learned discrete set, and can be seen as a cluster centroid of the latent speech representation space, the cluster including $\mathbf{z}_t$. It is this $\mathbf{q}_t$ the context network is trained to predict. This is an additional intrinsic bias to learn phonetic units, as speech naturally clusters into acoustic units related to phonemes.

We will follow with a detailed description of the quantization process, the masking strategy, and the objective function(s) of the pretext task.

Quantization

Each $\mathbf{z}_t$ in a sequence is individually classified to a quantized vector $\mathbf{q}_t$, thereby creating $\mathbf{Q} = \mathbf{q}_1, \ldots, \mathbf{q}_T$. The possible quantized vectors are learned, and represented by *codebook*s, discrete sets of real-valued vectors. A codebook G is a set of V entries (vectors, or codewords) of a particular dimension d_G, representable as a matrix of size $V \times d_G$. A single linear layer with softmax activation can be used to get a probability distribution over V different classes. The class with maximum probability can then be used to determine $\mathbf{q}_t$ from $\mathbf{z}_t$. To (efficiently) increase the number of possible vectors, multiple codebooks and linear layers can be used, where the final quantized vector is the concatenation of the classification result from each codebook. In the `BASE` configuration of wav2vec 2.0 there are two codebooks with each $V = 320$ entries of size $d_G = 128$, resulting in $320^2 = 102\,400$ quantized vectors with dimensionality $d_q = 256$. In the `LARGE` configuration $d_G = 384$, resulting in $d_q = 768$.

However, the procedure above is not differentiable due to the selection operation. This is circumvented by using the gumbel-softmax operation (Jang et al., 2017) after the linear layer, instead of using argmax on the softmax output. The gumbel-softmax returns a one-hot logits vector during the forward pass, which implies that the discrete

vector can be (differentially) selected with a weighted sum of the one-hot logits over the entries of V in G. Additionally, like softmax, the gumbel-softmax can be controlled with a temperature parameter τ. At the start of training, $\tau = 2$, which makes the gradient of codebook entries more uniform. This is gradually decreased to $\tau = 0.5$ with a step-wise factor of 0.999995 during training, which makes the gradient of the non-selected codebook entries smaller.

Masking

The masking is done in the context network, after the normalization and projection, but before computing the relative positional embedding. The mask consists of multiple regions of $L_m = 10$ consecutive latent speech vectors which are all replaced by the same (learned) mask vector. In total $p_m = 50\%$ of the latent vector sequence Z are masked, with the possibility that some regions overlap. The set of time steps where masking is applied is indicated by $\mathbf{M}$. Formally, we write $\hat{\mathbf{z}}_t = m(\mathbf{z}_t)$, where $m(\cdot)$ is the masking function. The number of mask regions for a given length T is $n_r = \lfloor T \frac{p_m}{L_m} \rfloor$. The length T is excluding potential padding vectors, in the case a batch of utterances have different length. The starting positions of the masked regions is determined by randomly choosing $n_r = \lfloor \frac{T}{20} \rfloor$ distinct time indices in the range $1, \ldots, T$.

Objective function

The objective function during self-supervised pre-training consists of a weighted sum of the main *contrastive* loss $\mathcal{L}_c$, together with an auxiliary *diversity* loss $\mathcal{L}_d$ with weighing λ_d, and an auxiliary L^2*-penalty* loss $\mathcal{L}_p$ with weighting λ_p:

$$\mathcal{L}_{\text{ssl}} = \mathcal{L}_c + \lambda_d \mathcal{L}_d + \lambda_p \mathcal{L}_p$$

Contrastive loss

The contrastive loss $\mathcal{L}_c$ encapsulates the pretext task, where we use the masked $\hat{\mathbf{Z}}'$ as input to the transformer in $g(\cdot)$, which has to predict the cluster centroids $\mathbf{Q}$ of the masked values in the output $\mathbf{C}$. This prediction is done contrastively, such that for a given $\mathbf{q}_t$, the predicted value at $\mathbf{c}_t$ needs to be as similar as possible. At the same time, $\mathbf{c}_t$ needs to be as dissimilar as possible to time steps in $\mathbf{Q} \setminus \{q_t\}$. Dissimilarity is encouraged by sampling k *distractors* from $\mathbf{Q}$. The network is explicitly penalized if $\mathbf{c}_t$ is similar to any distractors sampled from $\mathbf{Q}$. Similarity is measured with the cosine similarity, written as $\mathbf{sim}(\mathbf{a}, \mathbf{b})$. The loss can then be defined as

$$\mathcal{L}_c(\mathbf{C}, \mathbf{Q}, \mathbf{M}) = \sum_{t \in \mathbf{M}} -\log \left(\frac{\exp\left(\frac{\mathbf{sim}(\mathbf{c}'_t, \mathbf{q}'_t)}{\tau_c} \right)}{\sum_{d \in D_t \cup \{t\}} \exp\left(\frac{\mathbf{sim}(\mathbf{c}'_t, \mathbf{q}'_d)}{\tau_c} \right)} \right)$$

where D_t is random sample of k values from $\mathbf{M} \setminus \{t\}$. Note that at any time the current time step t is excluded from the sampling. The values $\mathbf{c}_t$ and $\mathbf{q}_t$, with dimensionality $d_c = 768$ and $d_q = 256$ in the `BASE` configuration, are respectively projected to $\mathbf{c}_{t'}$ and $\mathbf{q}_{t'}$, both with dimension $d_{\text{sim}} = d_q$ so that the cosine similarity can be computed. A temperature $\tau_c = 0.1$ leads to a hard softmax distribution, controlling the gradient to focus on making correct predictions more than being dissimilar to distractors. The contrastive loss can be interpreted as a standard $1 + k$ classification task with the cross-entropy criterion. The logits are provided by sim, softmax is applied over the logits, and the target is always the class index representing q_t.

Diversity loss

A shortcut to optimizing the contrastive loss is to map all values in $\mathbf{Z}$ to the same quantized vector. To prevent this, a diversity loss is applied, which encourages uniform predictions over the codebook entries. A codebook G with V entries has classified the sequence $\mathbf{Z}$ to $\mathbf{Q}$, using logits from a softmax activations of a linear layer $\mathbf{P} = \mathbf{p}_1, \ldots, \mathbf{p}_T$. For uniform predictions the average probability distribution

$$\tilde{\mathbf{p}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{p}_t$$

should be flat. In this *best* case, the entropy $H(\tilde{\mathbf{p}}) = \log V$, and the perplexity $e^{H(\tilde{\mathbf{p}})} = V$. In the *shortcut* case, a single class has probability 1, which means the entropy $H(\tilde{\mathbf{p}}) = 0$ and the perplexity 1. Therefore, to penalize a shortcut case, and hopefully preventing it from occurring, the diversity loss minimizes the number of the entries in a codebook subtracted by the perplexity of the predictions:

$$\mathcal{L}_d(\mathbf{P}) = V - \exp\left(-\sum_{j=1}^{d_q} \tilde{p}^{(j)} \log \tilde{p}^{(j)} \right)$$

where $\tilde{p}^{(j)}$ is the jth component of $\tilde{\mathbf{p}}$. Note that for the diversity loss, the logits of the linear layer are activated with vanilla softmax, while the selection of Q is done by activating the logits with the gumbel softmax (including a temperature τ). In practise, there can be multiple codebooks, in our case we have G_1 and G_2, with equal number of entries V and dimension d_G. We compute the loss separately for both codebooks and sum the result. The weighting $\lambda_d = \frac{1}{10}$ throughout this work.

L^2-penalty loss

The third loss is a regularization term, which keeps the values of $\mathbf{Z}$ as small as possible. This loss is defined as

$$\mathcal{L}_p(\mathbf{Z}) = \frac{1}{T d_z} \sum_{t=1}^{T} \sum_{j=1}^{d_z} \left(z_t^{(j)} \right)^2$$

where $z_t^{(j)}$ is the jth component of $\mathbf{z}_t$, and $d_z = 512$. The weighting $\lambda_p = 10$ throughout this work.

5.2.3 Batch creation

The methodology description so far has assumed a single utterance $\mathbf{X}$, while training is done with a batch of the dataset, split into multiple gpu-batches for distributed data-parallel training. For pre-training we mainly use the Librispeech dataset, implying utterances have variable lengths, at minimum 0.83 seconds, and at most 30 seconds. As each utterance in a gpu-batch needs to have the same length, all but the longest raw waveform in a batch are padded with zeros. To minimize the amount of padding, the utterances are sorted by length, and put into bins of 5000 utterances. Each gpu-batch is sampled from only a single bin. Random samples from the bin feed a priority queue of length 300, from which gpu-batches are formed by taking samples prioritized by shortest *or* longest duration, whichever results in the fewest padding, until the total speech duration in the gpu-batch exceeds a threshold, in our case 2.4 M samples. Because of limitation in GPU memory caching, a gpu-batch is discarded if the difference between the shortest and longest utterance is more than 10 seconds. This helped alleviate spontaneous CUDA out-of-memory errors.

The CNN also processes the padded part of utterances. However, all time steps with a $\mathbf{z}_t$ which results purely from padding in the raw waveform are ignored in the self-attention of the transformer by setting their attention score to $-\infty$. When creating the mask $\mathbf{M}$, the padded vectors $\mathbf{z}_t$ are also not considered part of the utterance. The contrastive loss is computed independently for each masked token in each utterance of the gpu-batch, and summed afterwards. For the diversity loss, the probability distribution $\tilde{\mathbf{p}}$ is computed by averaging over the predictions of all tokens of all utterances in the gpu-batch, before computing the perplexity. The L^2-penalty loss is simply the mean of each representation value in the gpu-batch. The gradient resulting from each gpu-batch are averaged before the weights of the networks are updated.

5.2.4 Fine-tuning for ASR with varying amount of labels

To (fully) fine-tune a pre-trained model for speech recognition, $\mathbf{Z}$ and $\mathbf{C}$ can be computed from $\mathbf{X}$, disregarding the quantization. The network still applies a mask, but only $p_m = 5\%$ of the utterance is replaced with the learned masking vector. This acts as a regularization method, similar to SpecAugment (Park et al., 2019). Each vector $\mathbf{c}_i \in \mathbf{C}$ can be separately classified to a character (or blank) with a softmax-activated linear layer, and optimized with CTC loss (Graves et al., 2006). The CNN is not updated during fine-tuning, and the transformer network is only updated after the first 5 k iterations. We fine-tune on 10 min, 1 hour, 10 hours, 100 hours and 960 hours of Librispeech following (Baevski et al., 2020a).

5.2.5 Frozen fine-tuning using the SUPERB benchmark

Another fine-tuning strategy is used in the SUPERB benchmark (Yang et al., 2021), where the (upstream) CNN and transformer layers will be frozen and used only to generate input features for a task-dependent, small downstream model, e.g., a 2-layer biLSTM for speech recognition, or a single dense layer followed by mean pooling for speaker identification. The input features $\mathbf{F} = \mathbf{f}_1, \ldots, \mathbf{f}_T$ are a weighted-sum of $\mathbf{Z}'$ and $\mathbf{C}$, i.e., $\mathbf{f}_t = w_0 \mathbf{z}'_t + \sum_{i=1}^{I} w_i \mathbf{c}_t^{\{i\}}$, where $^{\{i\}}$ indicates the output sequence of the ith transformer layer. This is in contrast with pre-training and full fine-tuning, where only the last output sequence is considered, i.e., $I = 12$ for `BASE` or $I = 24$ for `LARGE`. The weights w_i are learned during fine-tuning. We use a subset of tasks in (Yang et al., 2021) to limit computational resources, while including at least one task from 4 out of the 5 categories: phoneme recognition (content), ASR in English (content), out-of-distribution ASR in Mandarin (content), speaker verification (speaker), emotion recognition (prosody), and intent classification (semantics). We use the default downstream model for each task, and keep most settings to the default value. We make the following modifications to reduce the run-time: half- instead of single-precision floats, 200 k instead of 500 k train steps for Mandarin ASR, and gradient accumulation of 1 for all tasks, while increasing the batch size such that the effective batch size is equal to the default settings. A constant learning rate of 10^{-4} is used for all tasks.

5.3 Experiments

5.3.1 Pre-training with different batch sizes

The first experiment aims to directly answer RQ 1, and is a prerequisite for answering all others.

Setup

We pre-train the `BASE` wav2vec 2.0 network with batch sizes ranging from 87.5 seconds to 80 minutes of audio, as seen in Table 11. Each pre-training starts with the same initial weights, and we use all 960 hours of training data in Librispeech (Panayotov et al., 2015), with 5 % held-out randomly as a validation set. We validate and store a checkpoint every 5 k steps. We adhere to the hyperparameters as published in the seminal paper (Baevski et al., 2020a) as much as possible. We use 400 k training iterations with AdamW (Loshchilov and Hutter, 2019), a weight decay of 10^{-2}, and scan over 3 learning rates (LRs), based on choices explained below. We change from a tri-stage LR schedule to a 8-cycle triangular LR schedule, where one cycle has 25 k linear steps up and 25 k linear steps down. This allows us to fairly compare fine-tuning results of checkpoints at multiples of 50 k iterations. The minimum LR of the cycle is 100 times smaller than the maximum LR. We show the maximum LRs in Table 11. We also use GPUs (A5000) with at least 24 GB of VRAM , and therefore fill each GPU with a maximum of 2.4 M audio samples (150 seconds) for full utilization of the

Table 11: *All batch sizes used for SSL pre-training, together with the number of GPUs, the number of gradient accumulation steps (acc.), the runtime in days and hours of a single run, and three possible learning rate heuristics. The bold learning rates resulted in the lowest validation loss. For the 80-minute batch size setting, we only tried one learning rate.*

batch size					learning rate heuristics		
sec	min	GPUs	acc.	runtime	h_{const}	h_{sqrt}	h_{lin}
87.5	1.5	1	1	1d 13h	$5.00 \cdot 10^{-4}$	$\mathbf{6.04 \cdot 10^{-5}}$	$7.29 \cdot 10^{-6}$
150	2.5	1	1	2d 4h	$5.00 \cdot 10^{-4}$	$\mathbf{7.91 \cdot 10^{-5}}$	$1.25 \cdot 10^{-5}$
300	5	1	2	3d 22h	$5.00 \cdot 10^{-4}$	$\mathbf{1.12 \cdot 10^{-4}}$	$2.50 \cdot 10^{-5}$
600	10	4	1	2d 14h	$5.00 \cdot 10^{-4}$	$\mathbf{1.58 \cdot 10^{-4}}$	$5.00 \cdot 10^{-5}$
1200	20	2	4	7d 20h	$5.00 \cdot 10^{-4}$	$\mathbf{2.24 \cdot 10^{-4}}$	$1.00 \cdot 10^{-4}$
2400	40	4	4	7d 18h	$\mathbf{5.00 \cdot 10^{-4}}$	$3.16 \cdot 10^{-4}$	$2.00 \cdot 10^{-4}$
4800	80	8	4	7d 19h	$\mathbf{5.00 \cdot 10^{-4}}$	-	-

device, which compares to 1.4 M samples (87.5 seconds, 90 minutes batch size with 64 GPUs) in (Baevski et al., 2020a). The experiments, including initial development runs, took 246 days of GPU time.

For each batch size of duration s, we need to find a well-performing maximum learning rate for the cyclic schedule. As a full hyperparameter search would exceed our computational budget, we use heuristics to choose three different learning rates, and settle on a run with the lowest overall validation loss. The first heuristic is the linear scaling law from (Goyal et al., 2018). As a reference, a maximum learning rate of $m_{\text{lr}} = 5 \cdot 10^{-4}$ was used in (Baevski et al., 2020a) together with a batch size of circa 1.6 hours. Therefore we use

$$h_{\text{lin}(s)} = m_{\text{lr}} \frac{s}{s_{\text{orig}}}$$

as the first heuristic for the learning rate, with $s_{\text{orig}} = 6000$ seconds. In (Baevski et al., 2020a) a batch size of 5600 seconds is used, but we use 6000 seconds for this heuristic calculation so that h_{lin} rounds nicely. We still find well-performing LRs. The second heuristic

$$h_{\text{sqrt}(s)} = m_{\text{lr}} \sqrt{\frac{s}{s_{\text{orig}}}}$$

applies the square root scaling law according to (Malladi et al., 2022). For each batch size we also try the constant

$$h_{\text{const}(s)} = m_{\text{lr}}$$

although this led to divergence for $s \leq 600$ seconds.

Moreover, we initially used a diversity loss weight $\lambda_d = 0.1$ as in (Baevski et al., 2020a), but found this led to divergence of the diversity loss for the batch size of 87.5 seconds. By decreasing λ_d from 0.1 to 0.05, with the reasoning that the contrastive loss is almost twice as small (only 1.4 M samples instead of 2.4 M), and the ratio between the contrastive loss and the diversity loss should stay the same, we managed to get converging results for the batch size of 87.5 seconds. This does seem counter-intuitive as the original work uses 1.4 M samples on each of the 64 GPUs together with $\lambda_d = 0.1$.

Results

First, we show various metrics during the training procedure in Figure 5. For each batch size, the metrics of the run with the lowest validation loss are displayed. For the contrastive loss (A), we see that overall a *larger* batch size leads to a lower contrastive loss. Note that the smallest batch size, 87.5 seconds, has a different range because the loss is sum-reduced over 1.4 M sampled instead of 2.4 M samples, we corrected for this by doubling the values. For the diversity loss (B), we see that a large batch size (40, 80 min) causes the loss to drop quickly, but then plateau. The other batch sizes steadily decrease. Notably, for batch sizes of 10 and 20 minutes the diversity loss surpassed the values of batch sizes 40 and 80 minutes after 150 k to 200 k steps. The scale of the lowest batch size, 87.5 seconds, is halved, as $\lambda_d = 0.05$. We corrected for this by multiplying the values by 2. The same patterns visible in the diversity loss are also seen in the perplexity of the codebooks (H and I, with the vertical scale reversed). Without exceptions, for the L^2-penalty loss (C), larger batch sizes lead to a higher loss. For accuracy (D), a larger batch size leads to higher accuracies. There are minor differences between the perplexity of the codebooks. Note that the numbering of the the codebooks is arbitrary. We decided that "codebook 1" generates the top half of the quantized vectors $\mathbf{q}_i \in \mathbf{Q}$, and "codebook 2" generates the bottom half. The two codebooks are initialized differently compared to each other, but these initializations are constant for all training runs. First, we note that the perplexity values are slightly lower in codebook 2. Secondly, the perplexity of a 10 min batch size grows larger in codebook 1 than codebook 2. Also, in codebook 1, the perplexity of 80 min batch size is larger than 40 min batch size, but in codebook 2, this is the other way around.

Secondly, we plot the similarity between codewords within the codebooks for each batch size in Figure 6. We see that the average (A, B) and minimum (C, D) cosine similarity between codewords only goes down steeply with large batch sizes (40 and 80 min). Additionally, for batch sizes of 10 and 20 minutes, we observe a lower minimum similarity over the training procedure. For the maximum similarity values, we observe that they stay relatively stable, although for the larger batch sizes they increase slightly at the start of training, but decrease again during the training procedure, which can be related to the decay strategy of τ used in the gumbel-softmax.

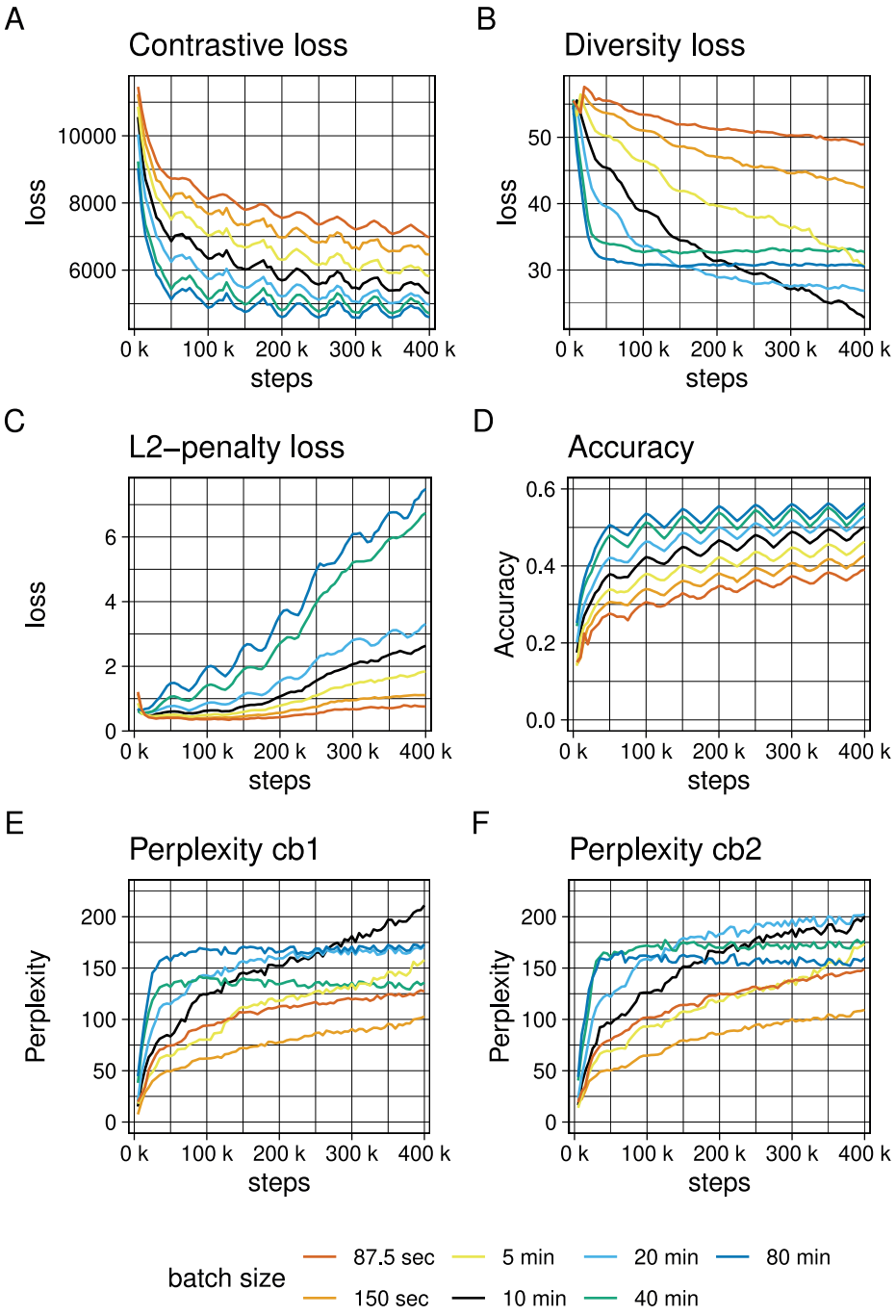


Figure 5: *Various metrics on validation data (interval of 5 k training steps) during self-supervised pre-training with different batch sizes, namely all three losses (A, B, C), the accuracy of predicting the correct masked quantized vector (D), and the perplexity of codebook 1 (E) and codebook 2 (F).*

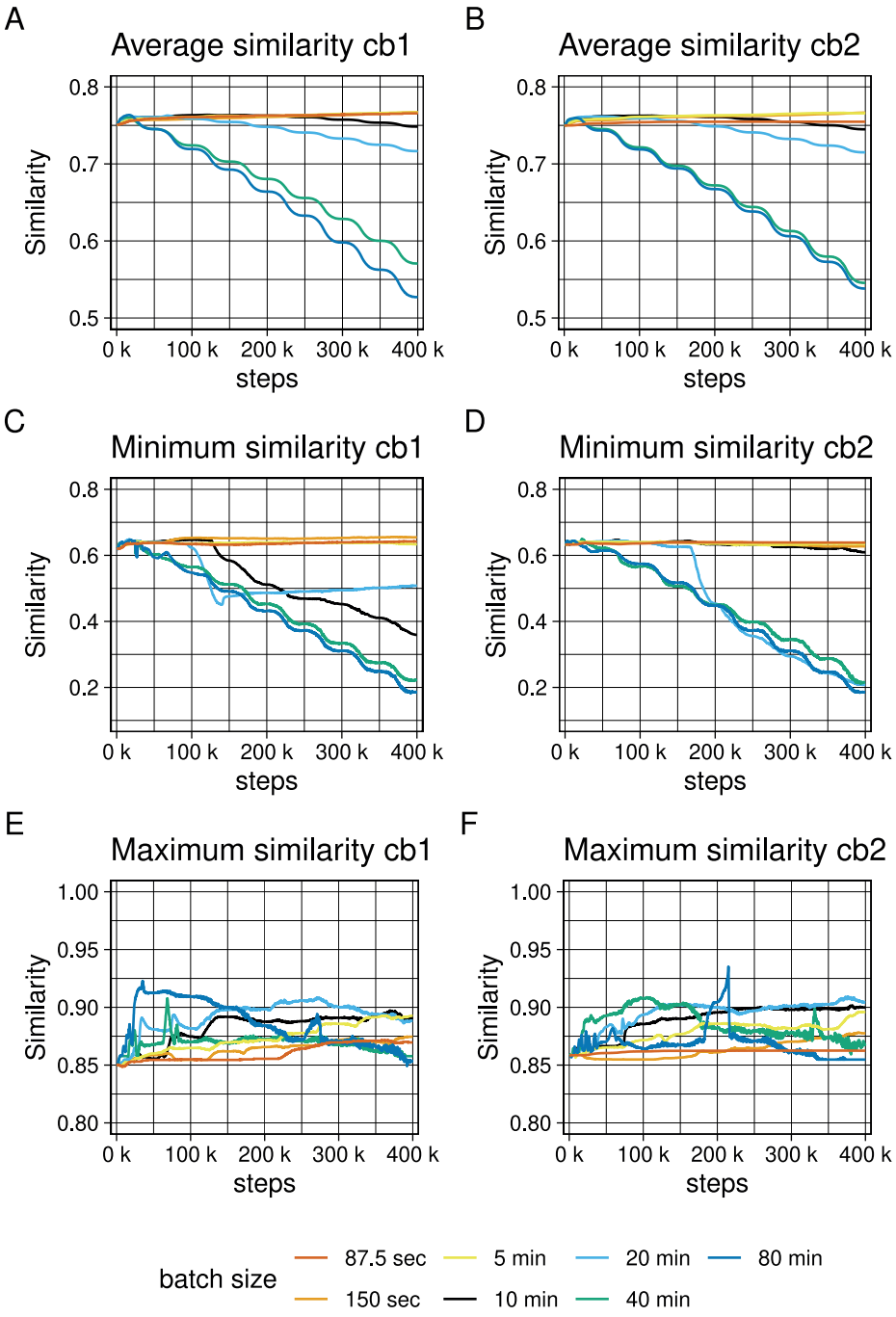


Figure 6: *The average, minimum, and maximum value of the pair-wise cosine similarity of all codewords in codebook 1 (A, C, E) and codebook 2 (B, D, F) during self-supervised pre-training for various batch sizes, with an interval of 100 steps.*

5.3.2 ASR fine-tuning with varying amounts of labels

The second experiment focuses on RQ 2. How is downstream fine-tuning affected by the batch size during pre-training?

Setup

For each batch size in Table 11 we have self-supervised training runs of 400 k steps, with checkpoints saved every 5 k steps. For each setting we select the run (and step) with the lowest overall validation loss, resulting in a single checkpoint which is used as initialization for training a speech recognition system. This is the checkpoint at step 400 k for all runs, expect for batch size of 32 GPUs (80 minutes), which had the lowest validation loss at step 305 k. For each of these selected checkpoints we perform a fine-tuning on 10 minutes, 1 hour, 10 hours, 100 hours, and 960 hours of labeled Librispeech data, with hyperparameters detailed below. We use the same number of steps as in (Baevski et al., 2020a). We show results with greedy letter decoding and word decoding using a 4-gram Librispeech language model. For word decoding we use a beam size and threshold of 50, a language model weight of 2, and a word insertion score of 0 for all settings. These experiments were done using one A5000 GPU, with a maximum run-time of 2 days when fine-tuning with 960h of labels (320k steps). In total 205 days of GPU time was used, including experiments in Section 5.3.4.

We use similar fine-tuning parameters for each labeled data condition as stated in (Baevski et al., 2020a), but make some changes such that the only variation is in the number of iterations. We use 12 k, 13 k, 20 k, 80 k and 320 k iterations, respectively for 10 minutes, 1 hour, 10 hours, 100 hours, and 960 hours of labeled fine-tuning data. We use a learning rate of $5 \cdot 10^{-5}$ with Adam, not using weight decay. We use the same tri-stage learning rate schedule, where the first 10% of iterations warm up the LR linearly from $5 \cdot 10^{-7}$ to $5 \cdot 10^{-5}$, the next 40 % of iterations keep the LR constant at $5 \cdot 10^{-5}$, and the last 50 % of iterations exponentially decay the learning rate from $5 \cdot 10^{-5}$ to $2.5 \cdot 10^{-6}$. We fine-tune with a single GPU, using a batch size of 3.2 M samples (200 seconds). The CNN network is frozen for all iterations, while the Transformer network is frozen for the first 5 k iterations. Masking is applied on the $\mathbf{Z}$ sequence, but only 5 % of the sequence is masked. We do not apply masking on the feature dimension. We also do not use LayerDrop, as we did not use LayerDrop during pre-training to simplify data-parallelism. Dropout is set to 10 % in the Transformer layer (also during SSL).

Results

We show the word-error-rate (WER), evaluated on Librispeech `test-clean` and `test-other`, for each fine-tuning condition in Figure 7. Two clear patterns are visible. First, independent of the amount of labels available, we observe that fine-tuning a random initialization leads to the highest WER. Then, each consecutive increase in the batch size during self-supervised learning leads to lower WERs after fine-tuning.

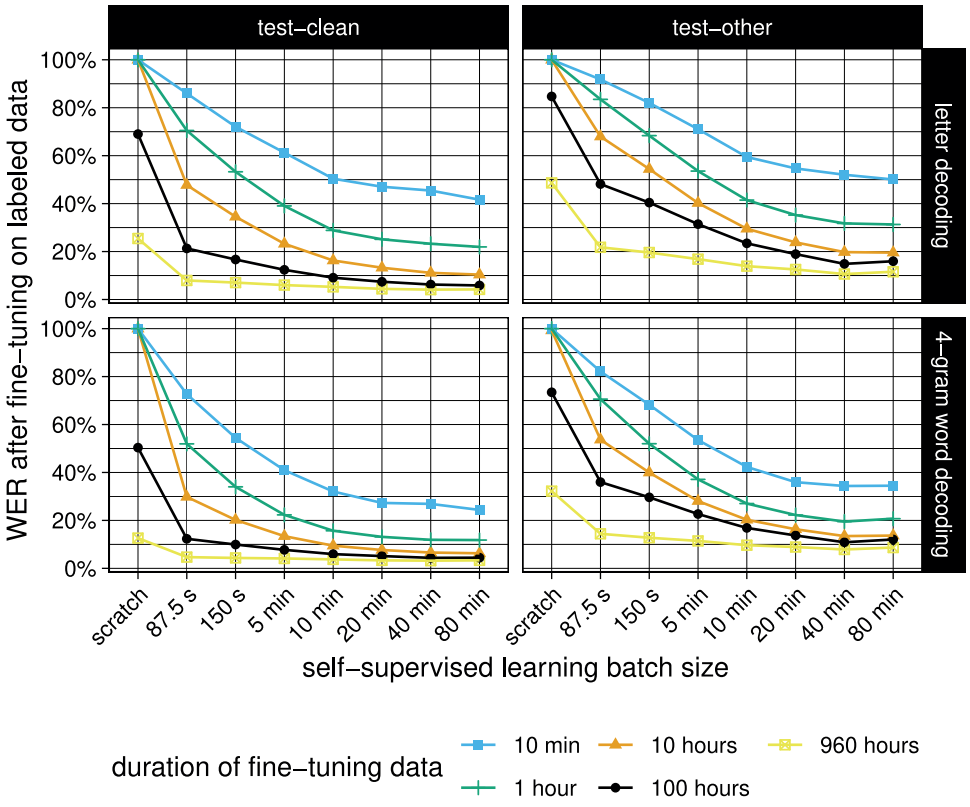


Figure 7: *The WER (left column: Librispeech* `test-clean`*, right column: Librispeech* `test-other`*) against the batch size during pre-training of a self-supervised initialization. The self-supervised models are fine-tuned for speech recognition using 5 different magnitudes of labeled data. Scratch indicates fine-tuning a random initialization instead of a self-supervised initialization. The upper row shows the WER with letter decoding, while the bottom row shows the WER with word decoding using a 4-gram language model.*

There is one exception: on test-other, the 40 min batch size initialization performs better than the 80 min batch size initialization, but only when fine-tuning with 10 or more hours of labeled data. We observed similar degraded performance after fine-tuning the 400 k checkpoint with batch size of 80 minutes (not shown in Figure 7). Secondly, having more labeled data for fine-tuning leads to a lower WER for each self-supervised batch size. However, the larger the batch size, the smaller the difference in WER between the amount of labels available during fine-tuning. Notably, (Baevski et al., 2020a) reports 9 % and 47% WER on `test-clean`, respectively with and without a language model, when fine-tuning with 10 minutes of labeled audio. In this experiment we observe a WER of respectively 24 % and 41 % instead. This large difference in LM

performance we attribute to our much smaller beam size in decoding. Finally, we see diminishing returns at a batch size of 80 min.

5.3.3 Analysis on effectiveness of large batch sizes

So far, we have observed that larger batch sizes lead to a lower contrastive validation loss, and less similarity between codewords. We have also seen that larger batches result in better fine-tuning performance for speech recognition, irrespective of the amount of labels available. Why are large batch sizes more effective? Are better gradients approximations beneficial, or is the amount of observed data an explaining factor?

Variance of gradients

First, we compare the gradient between different batch sizes. If the gradients are more precise, and less noisy, with increased batch sizes, we expect the variance between

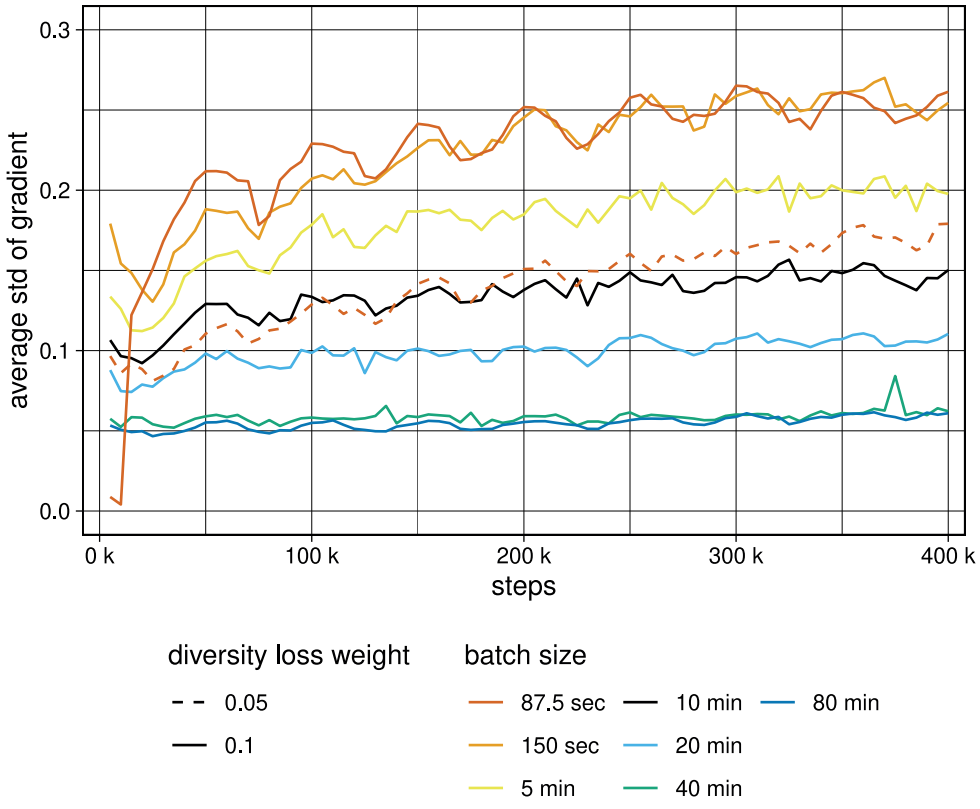


Figure 8: *The standard deviation of the gradient of $\mathcal{L}_{\text{ssl}}$ for each batch size, computed over 10 random batches, and averaged over all parameters, against consecutive checkpoints during pre-training.*

gradients to decrease. To verify this we use the saved checkpoints (every 5 k steps) during pre-training. For each checkpoint, we compute 10 gradient vectors using new, independently sampled batches from the training set. These 10 batches are kept constant over all checkpoints of the same batch size. We do not update the weights during this process, and we do not use the AdamW optimizer state nor the learning rate to scale the gradients. For each parameter, we separately compute the standard deviation between gradient vectors, after which we average over the parameters to get a standard deviation of the whole gradient, shown in Figure 8. We observe that the standard deviation decreases as the batch size increases. For the smallest batch size, the training run with $\lambda_d = \frac{1}{20}$ has a significantly lower gradient variance compared to $\lambda_d = \frac{1}{10}$. Furthermore, at a critical batch size of 40 minutes the standard deviation barely decreases when the batch size is doubled to 80 minutes. Furthermore, for small batch sizes the standard deviation increases over the training procedure, while it stays constant for large batch sizes. Moreover, we observe that the batch sizes of 87.5 (with $\lambda_d = \frac{1}{10}$) and 150 seconds converge at the end of training. Finally, the cyclic learning rate schedule seems to affect the gradient variance, as the standard deviation peeks for all batch sizes when the cycle is at the minimum learning rate (every interval of 50 k steps).

5.3.4 Observing specific amounts of data during pre-training

The second hypothesis on why large batch sizes are more effective, namely that the model can simply observe more hours of data in the same amount of steps, is related to RQ 3, which asks whether a model trained with half the batch size but twice the amount of steps has equivalent performance. We will try to answer RQ 3 by comparing the performance of batch sizes at different stages during pre-training. Because we use a cyclic learning rate, there are equivalences at each end of a cycle, namely at multiples of 50 k steps. Different batch sizes overlap on the amount data seen at particular checkpoints. For example, 16.7 k hours of data was observed with batch sizes of 150 sec, 5 min, 10 min, and 20 min respectively at 400 k, 200 k, 100 k, and 50 k steps. If less noisy gradient approximations are beneficial to learning, we expect a performance difference when we compare the fine-tuning performance of these checkpoints, in favor of larger batch sizes. However, if all that matters is observing more data, we should see no difference in performance.

Setup

For each batch size, we fine-tune the checkpoints with an interval of 50 k steps, resulting in 8 checkpoints per batch size. We use the training methodology as described in Section 5.3.2. For this experiment we focus on fine-tuning on 10 minutes and 100 hours of labeled data, with letter decoding, evaluating on the Librispeech `test-clean` evaluation dataset.

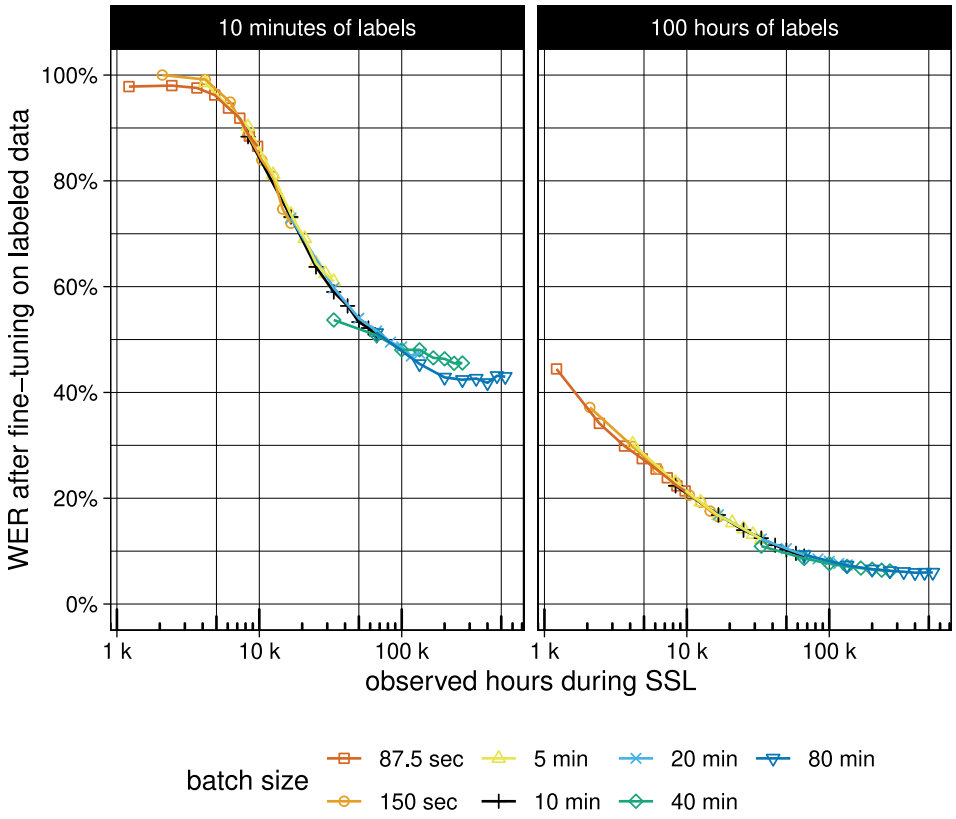


Figure 9: *The WER, after fine-tuning, against the hours of data processed during self-supervision.*

Results

The results are shown in Figure 9. We observe a direct relationship between the amount data observed during pre-traing and the WER after fine-tuning. There are only minor differences between the WER of different checkpoints with the same amount of data, which we attribute to noise. The curves for each batch size blend into each other, especially for the case of fine-tuning with 100 hours of data. With 10 minutes of data we observe that a batch size of 40 minutes has slightly better performance at the start of training, and worse performance at the end of training, compared to batches of 20 minutes and 80 minutes. Also, we see that a batch size of 87.5 seconds performs slightly better than the batch size of 150 seconds. We hypothesize that the diversity weight $\lambda_d = \frac{1}{20}$ generalized slightly better than $\lambda_d = \frac{1}{10}$. Note that we used the naive, upper bound of the amounts of data hours observed. Due to fact that audio samples in Librispeech are varied, batch sizes are filled up to a specific threshold, as explained in Section 5.2.3. The product of the batch size

Table 12: *The number of epochs and total amount of data observed throughout pre-training for each batch size. The training dataset contains 912 hours of data and we train for 400k iterations. Note that the batch size is an upper bound as they are constructed with variable length samples.*

batch size		upper bound		measured	
sec	min	epochs	hours	epochs	hours
87.5	1.5	11	10 k	11	9 k
150	2.5	18	17 k	18	16 k
300	5	37	33 k	36	31 k
600	10	73	67 k	71	62 k
1200	20	146	133 k	140	124 k
2400	40	292	267 k	277	248 k
4800	80	585	533 k	554	497 k

and number of iterations is therefore an upper bound on the amount and duration of samples observed, as the total size of a batch will rarely by equal to the threshold.[17] During pre-training, we stored the identifiers of every utterance in all 400 k batches. Afterwards, we could compute the actual amount of data observed, looking up the length of each utterance without padding. The results are shown in Table 12. We can see that the largest batch size has actually seen only 497 k hours of data, instead of the theoretical 533 k hours. This shows the importance of creating batches with as little length variability as possible, because our results have shown that seeing more hours of data in the same amount of iterations matters for downstream performance.

5.3.5 Fine-tuning various SUPERB benchmark tasks

To further strengthen the observation that the amount of hours seen during pre-training is the main indicator of downstream task performance, we repeat the experiment in Section Section 5.3.4 on 6 tasks in the SUPERB benchmark (Yang et al., 2021), namely phoneme recognition, speech recognition in English and Mandarin, speaker verification, emotion recognition, and intent classification. There are 3 important differences compared to the results shown in Figure 9.

1. We are interested to see whether the pattern holds for tasks other than (English) speech recognition.
2. The speech representation features are frozen during fine-tuning, so the quality of the representations are more fairly judged.

[17] Moreover, we discard some batches when the difference between the minimum and maximum file is larger than 10 seconds, as this prevented GPU out-of-memory errors. However, this is so infrequent (every end of an epoch) that the contribution of these skipped batches to the measured amount of observed data is negligible.

3. Other than phoneme recognition and English ASR, these tasks fine-tune on out-of-domain data with respect to the pre-training, so that we can see whether the observation holds in cross-domain adaptation settings.

Note that following datasets are used for fine-tuning and evaluation:

1. Librispeech (Panayotov et al., 2015) for phoneme recognition and English ASR.
2. Mozilla Common Voice (Ardila et al., 2020) for Mandarin ASR,
3. VoxCeleb1 (Nagrani et al., 2017) for speaker verification.
4. Fluent speech commands (Lugosch et al., 2019) for intent classification.
5. IEMOCAP (Busso et al., 2008) for emotion recognition.

Setup

For each of the 6 tasks, we fine-tune with 4 checkpoints of each SSL batch size, namely the checkpoint at 100 k, 200 k, 300 k and 400 k steps. We use a learning rate of 10^{-4} for each fine-tuning, and keep all other hyperparameters equal to the default value for the task, expect that mandarin ASR is fine-tuned for 200 k steps instead of 500 k steps.

Results

We show the results in Figure 10. In general, we observe similar patterns compared to Figure 9, meaning curves blending into each other, and following the pattern of better performance after seeing more data. We also see signs of overfitting with the batch size of 80 minutes at 400 k steps. We observe that the in-domain tasks (English ASR, and phoneme recognition) have the smoothest curve, followed by mandarin ASR. The other 3 tasks seem more noisy, where sometimes there is no improvement following a consecutive checkpoint of the same batch size, e.g., this is visible for speaker and emotion recognition at 10 k hours, and intent classification at 30 k hours of observed data during SSL. Also, emotion recognition is the only task where there is no clear upward trend with more observed data. For emotion recognition, the best performance was obtained with the checkpoint at 300 k steps with the batch size of 20 minutes, and the batch sizes of 40 and 80 minutes perform noticeably worse.

5.3.6 Increasing model capacity or changing pre-training dataset

All experiments so far have used the `BASE` wav2vec 2.0 network alongside pre-training on the 960 hour training split of Librispeech. In this section we will verify whether our observations generalize to a larger model capacity or a different pre-training dataset.

Setup

We pre-train with batch sizes of 5 minutes, 10 minutes, and 40 minutes of audio. To change the pre-training dataset, we use VoxCeleb2 (Chung et al., 2018) with wav2vec 2.0 `BASE`. The VoxCeleb2 dataset differs from Librispeech in that it is primarily designed for speaker recognition and created by automatically extracting

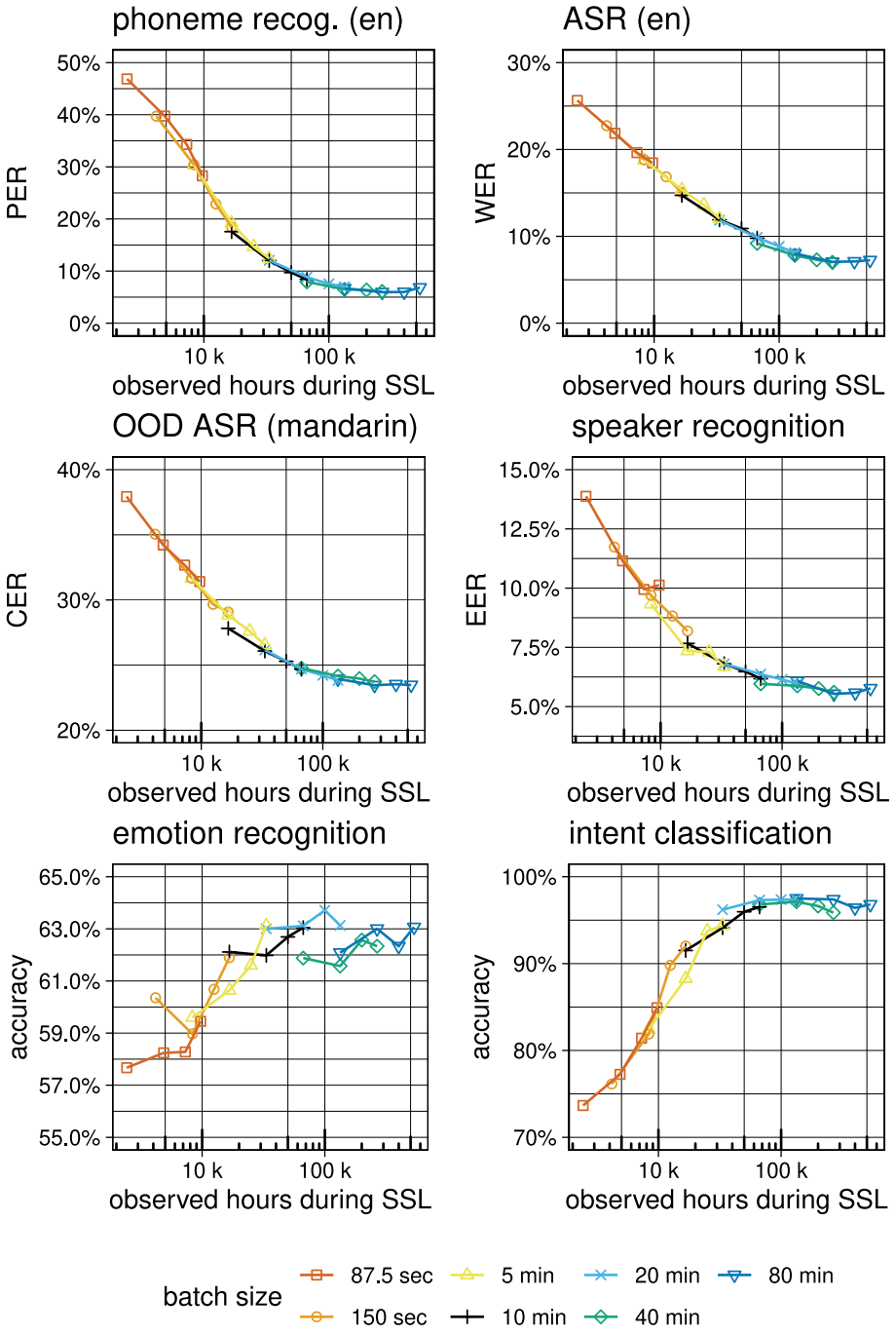


Figure 10: *The performance of 6 SUPERB tasks against the hours of data processed during self-supervision. For each SSL batch size we fine-tune the checkpoints at step 100 k, 200 k, 300 k and 400 k with learning rate* 10^{-4}.

utterances from YouTube videos. Compared to Librispeech it has more speakers (2338 versus 5994) and high intra-speaker variability. We split the VoxCeleb2 development set into a train and validation partition. The train partition has approximately 1 M utterances, with a duration ranging from 4 to 30 seconds averaging at 7.8 seconds, and a total of 2300 hours of audio. The validation partition is created by randomly selecting 2 utterances from each speaker session, and has a total of 23 hours of data. For pre-training we do not make any changes to the network or optimization procedure detailed in Section 5.3.1, other than performing a learning rate scan. For each batch size we perform a grid search in the range 10^{-4} to $7 \cdot 10^{-4}$ with increments of $5 \cdot 10^{-5}$.

To increase the model capacity, we use wav2vec 2.0 `LARGE` with Librispeech. The `LARGE` network has 24 transformer layers, with 16 heads, and a hidden dimensionality of 1024 which is scaled up to 4096 in the feed-forward network. Furthermore, the codeword dimensionality d_G is increased from 128 to 384, hence the dimensionality of the projection $\mathrm{sim}(c_t', q_d')$ in the contrastive loss is increased from 256 to 768. These modifications increase the amount of parameters from 95 M to 316 M. We also changed the floor of the gumbel softmax temperature in the quantization module from 0.5 to 0.1. Finally, we changed the diversity loss weighting from $\lambda_d = 0.1$ to $\lambda_d = 0.01$, because we observed that for all learning rates $\lambda_d = 0.1$ resulted in the diversity loss immediately converging and the contrastive loss never improving. We trained for 400 k steps with a batch size of 5 minutes and 250 k steps for batch sizes of 10 and 40 minutes. As in Section 5.3.1, we performed a learning rate scan for each batch size using the 3 heuristics h_{const}, h_{sqrt}, and h_{lin}. We used $m_{\mathrm{lr}} = 3 \cdot 10^{-4}$ and $s_{\mathrm{orig}} = 9600$ seconds taken from (Baevski et al., 2020a).

Results

For `BASE` with VoxCeleb2, we found the best performing LR for batch size 5, 10 and 40 minutes to be respectively $2 \cdot 10^{-4}$, $3 \cdot 10^{-4}$ and $5 \cdot 10^{-4}$, and for LARGE with Librispeech respectively $5.3 \cdot 10^{-5}$, $7.5 \cdot 10^{-5}$ and $1.5 \cdot 10^{-4}$. We fine-tune consecutive checkpoints (50 k steps for `LARGE`, and 100 k steps for VoxCeleb2) on the 6 SUPERB tasks. Each checkpoint is fine-tuned with learning rate 10^{-4}. We show the results with VoxCeleb2 as pre-training data in Figure 11 and for `LARGE` in Figure 12. For both settings we confirm the general trend of better performance as the visible training hours during pre-training increases. We observe for Mandarin speech recognition, and pre-training with VoxCeleb2, that the batch size of 40 minutes has slightly better performance across the amount of data observed, compared to the batch sizes of 5 and 10 minutes. For the speaker recognition task, we see for both VoxCeleb2 and `LARGE` that the EER does not monotonically decrease with the amount of hours of observed data within the optimization trajectory of a single batch size. This is consistent with observations in Figure 10. We see better EERs for speaker recognition with the `BASE` model pre-trained on in-domain data (VoxCeleb2) compared to the `BASE` and `LARGE` model pre-trained on out-of-domain data (Librispeech). Finally, for the phoneme

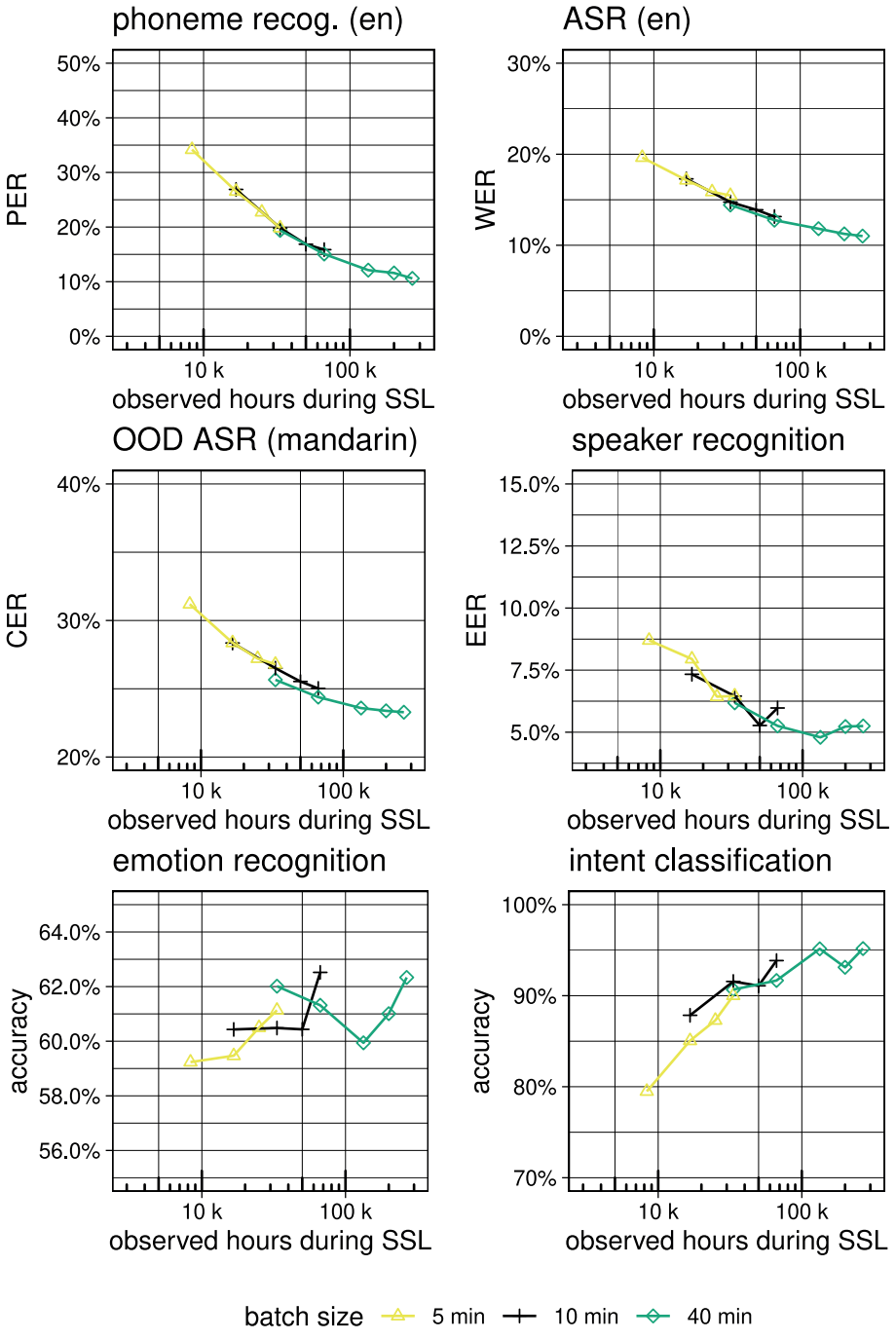


Figure 11: *The performance of 6 SUPERB tasks against the hours of data processed during self-supervision while pre-training on the VoxCeleb2 dataset. Each checkpoint is fine-tuned with learning rate* 10^{-4}.

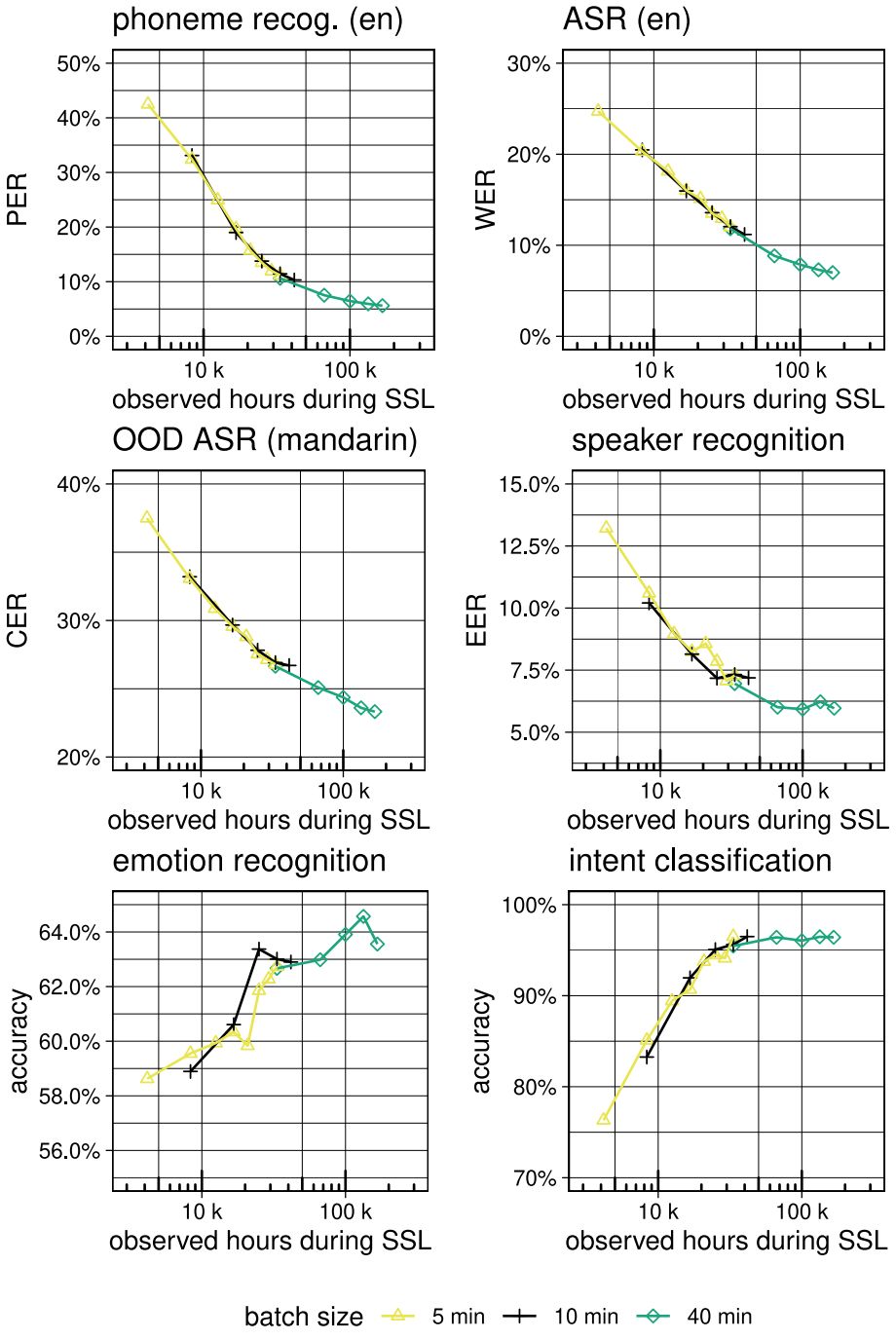


Figure 12: *The performance of 6 SUPERB tasks against the hours of data processed during self-supervision with the* `LARGE` *model. Each checkpoint is fine-tuned with learning rate* 10^{-4}.

recognition task, we see, for both conditions, the least amount of noise in the trend of performance versus the amount of data observed.

5.4 Discussion

Research questions

From the extensive search of batch sizes reported in Figure 5, we see that larger batch sizes result in better pre-training convergence, if given the same amount of iterations. This is consistent with the hypothesis of RQ 1 and RQ 2. We were surprised to observe convergence with all batch sizes, with the caveat that we had to change the diversity loss weighting for the smallest batch size of 87.5 seconds. It seems this problem with the loss weighting parameter choice could have been prevented if the contrastive loss would be mean-reduced instead of sum-reduced, but in this work we followed the implementation of (Baevski et al., 2020a) as closely as possible. A good indicator for well chosen hyperparameters is a continuous increase of the perplexity of the codebook logits (Figure 5 E–F). With larger batch sizes, the similarity of codebook vectors decreases, as seen in Figure 6, which is an indication of the diversity of the learnt representations.

Regarding RQ 2 and Figure 7, we show, for the first time, how pre-training batch size affects the downstream ASR performance: with a fixed number of iterations, the performance increases with larger batch size. All results with the 80 min batch size are in accordance with the original paper (Baevski et al., 2020a), except for the 10 minute fine-tuning results, where they decoded using the (impractical) beam size of 500. Our cyclic LR schedule does not perform worse than (Baevski et al., 2020a), while allowing for a fair comparison when conducting fine-tuning experiments at regular intervals. The largest batch size we investigated showed a little worse performance. We might have reached a limit of the generalization ability of the pretext task, as corroborated by the minimum validation loss at 305 k steps. We expect this limit can be increased by more regularization, e.g., using a larger data set (Conneau et al., 2021; Radford et al., 2023), or, to a lesser extent, higher dropout rates or higher value of λ_p and p_m.

In looking for an answer to why larger batch sizes are more effective, we saw in Figure 8 that the standard deviation of the gradients reduces almost consistently with larger batch size, up to a value of 40 min. However, we do not see an indication that this has an effect on downstream task performance, cf. Figure 9 and Figure 10. The reduced variance in the gradient as a result of a larger batch size does not appear to be the reason for better performance. Rather, we found that the most important factor for downstream task performance is the total amount of data seen during pre-training, i.e., the product of batch size and number of iterations (RQ 3), as shown convincingly in Figure 9, Figure 10, Figure 11 and Figure 12. This means that it is still possible to carry out pre-training with limited amount of GPUs and/or memory, but one needs to be more patient, or accept a penalty in performance, where Figure 7 can help in decision making.

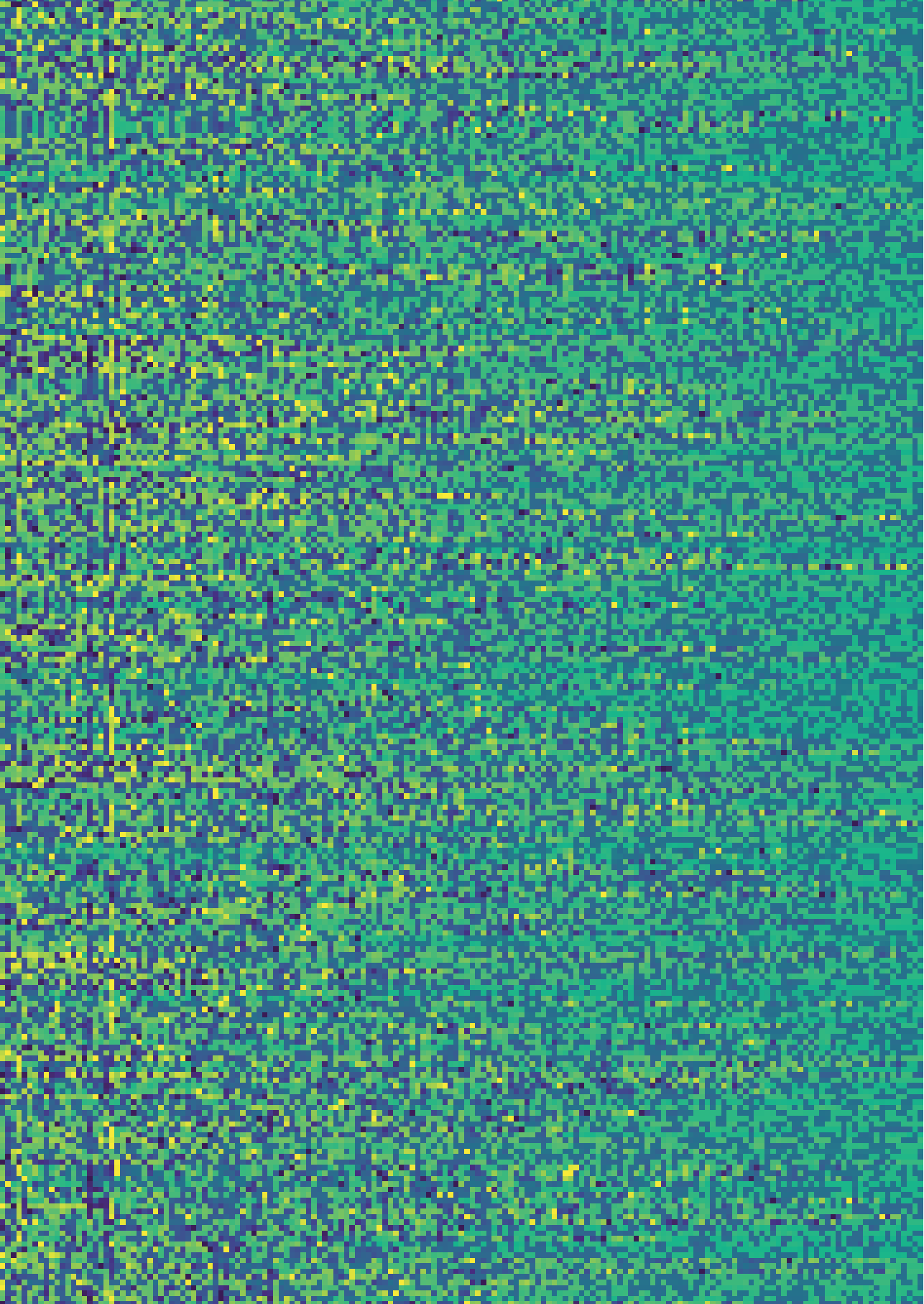
For the non-content tasks in Figure 10, namely speaker, emotion and intent recognition, there appears to be more noise in relation between amount of data seen and performance. This noise is strongest for emotion recognition, however, the results lie in a 10 % accuracy bandwidth, which is similar to the range of worst to best systems in the SUPERB leaderboard (Yang et al., 2021). The small size of the IEMOCAP dataset (10 speakers with 5-fold cross validation) probably adds to the noisy behavior. We believe this task may not be the best to indicate the quality of the learned speech representations. When the pre-training dataset is out-of-domain with respect to the fine-tuning dataset, we also see more noisy behavior, cf. Figure 11, however, the general trend still is visible. Finally, we see that the trend also holds when we increase the model capacity, cf. Figure 12.

Broader impact

Based on these results, we believe that it could benefit the community to benchmark SSL algorithms (in speech) by constraining the amount of data seen in training, e.g., to 100 k hours. In experiments with different algorithms, one might use 10 k hours of seen data to reduce the computational burden, and verify conclusions at the 100 k hours pre-training condition.

5.5 Conclusions

We conclude that the batch size during contrastive pre-training can be varied over a large range of values without a performance penalty, but only if hyperparameters like the learning rate are adapted accordingly. As a caveat, our results have only looked at contrastive algorithms where distractors are *not* taken from other samples in the batch. Future work could look at algorithms where this *is* the case, such as SimCLR (Chen et al., 2020), or a modified wav2vec 2.0 (Baevski et al., 2020a), paying special attention to the square root scaling law of the learning rate (Malladi et al., 2022), so that all batch sizes have similar well-performing optimization trajectories. Moreover, other speech representation learning algorithms could be considered, such as predictive methods like HuBERT (Hsu et al., 2021), WavLM (Chen et al., 2022a) and DinoSR (Liu et al., 2023), and reconstructive methods like VQ-VAE (van den Oord et al., 2017) and DeCoAR (Ling et al., 2020). The range of the batch sizes we found effective for wav2vec 2.0 may be specific to the architecture (wav2vec 2.0 `BASE`, with approximately 95 M parameters) and our pre-training datasets, but we have shown that also a larger model, and a different pre-training dataset, show a dependence on the amount of data seen, as seen in Figure 11 and Figure 12, in terms of the fine-tuning performance.



6 Self-supervised learning of speech representations with Dutch archival data

In which our adventurer considers the quality requirements for a pre-training dataset and uncovers the benefits of pre-training on a large-scale, mono-lingual dataset.[18]

Speech foundation models, such as wav2vec 2.0 (Baevski et al., 2020a), HuBERT (Hsu et al., 2021), and WavLM (Chen et al., 2022a), see widespread usage in various speech technology tasks (Yang et al., 2021). While these models differ in their self-supervised learning (SSL) objective function, they have in common that they are all developed and tuned with clean, pre-processed, labeled dataset(s), namely Librispeech (Panayotov et al., 2015) and Libri-light (Kahn et al., 2020) for wav2vec 2.0 and HuBERT, while WavLM also uses GigaSpeech[19] (Chen et al., 2021) and VoxPopuli (Wang et al., 2021). The nature of these datasets is predominantly prepared speech, with varying microphone quality, quiet recording conditions, known (pseudo-anonymous) speaker labels, and the knowledge that a full recording often contains a single speaker. Even though the dataset labels are not used during SSL pre-training, they aid in dataset pre-processing (especially single-speaker recordings), and we argue they cause implicit assumptions on the data SSL algorithms require to work well.

One particular use-case of self-supervised learning is the paradigm of multi-lingual pre-training, followed by fine-tuning for a low-resource language, as introduced by the wav2vec 2.0 XLSR model (Conneau et al., 2021), but also relevant (with caveats) to the design of Whisper (Radford et al., 2023). Also for the Dutch language, this paradigm of a fine-tuning (XLSR), or directly using (Whisper), a multi-lingual model, is a popular approach (Wang and Van Hamme, 2023; Bălan et al., 2024). This is partially due to the fact that, to our knowledge, there is only one foundation model specifically designed for the Dutch language, released by Conneau et al. (2021). In this work, we want to find out whether there is an inherent benefit to multi-lingual

[18]This chapter is based on the publication Vaessen, N., and van Leeuwen, D. A. (2025). Self-supervised learning of speech representations with Dutch archival data., in *INTERSPEECH 2025.*.

[19]The authors of WavLM specifically state they use a 10 k hour subset of the 40 k hour GigaSpeech dataset which is validated to not have utterances containing silence or noise.

pre-training, or whether, given the same computational budget and dataset size, a Dutch mono-lingual pre-training is more beneficial for Dutch speech recognition.

In order to answer this question, we construct a 55 k hour Dutch audio dataset for pre-training. For this purpose, we made use of the collection of archival Dutch television broadcast data from the Netherlands Institute for Sound and Vision. However, in order to create a high-quality pre-training dataset from this collection, we need to know the data quality required for convergence of a well-performing foundation model. Based on the datasets mentioned above, we speculate, first and foremost, that it needs to be easy to segment the dataset into sequences of audio with speech (utterances). These utterances must be relatively short, in the order of 1 to 30 seconds, and always contain speech for the full length of the utterance. Moreover, the speech in an utterance must be of a single speaker. Finally, there is limited background noise in the dataset, and no music is present, either instrumental, vocal, or a mix of both. Unfortunately, none of these conditions immediately apply to our collection of Dutch broadcast audio, which prompts us to perform an analysis on how to effectively clean our dataset.

Concretely, we ask the following research questions regarding self-supervised speech representation learning, and in particular, the contrastive wav2vec 2.0 algorithm, which is the only SSL algorithm we focus on due due to limited computational resources:

1. What constitutes a high quality dataset for self-supervised speech representation learning?
2. Can we construct such a dataset from television broadcast data?
3. Is pre-training on mono-lingual audio data better than pre-training on multi-lingual audio data?

Firstly, our hypothesis on high-quality datasets is that they need to be as similar to Librispeech as possible. We note that most self-supervised learning methods are solely developed on Librispeech-like data, simply due to ease of access and availability of read speech. However, this implies that SSL methods are overfit on a meta level, designed to work well with the particularities of Librispeech, but not on other kinds of datasets. To test the hypothesis, we will simulate various data quality scenarios by augmentation LibriSpeech with noise. Doing so will tell us what aspects are important for pre-processing our archival data. This neatly aligns with the second research question, where we assume that, by pre-processing our broadcast data correctly, it will be similar in quality to Librispeech, and therefore it can be used as a qualitative pre-training dataset. For the last research question, we will perform large-scale pre-training experiments with the pre-processing archival dataset. We hypothesize that, when equally sized, a mono-lingual dataset leads to higher-quality speech representations in that language, compared to a multi-lingual dataset, as more variation of that specific language can be observed, such as local dialects.

6.1 Related work

Ashihara et al. (2023) compare mono-lingual pre-training with Japanese data to fine-tuning multi-lingual pre-trained models, specifically, XLSR from Conneau et al. (2021). They show that a `BASE` wav2vec 2.0 model, pre-trained on a dataset of 500 hours of spontaneous Japanese speech, outperforms the `LARGE` XLSR model, when fine-tuned (50 hours) and evaluated (3.5 hours) on the Japanese newspaper reading dataset from Itou et al. (1999). Moreover, they show that pre-training on 200 hours of Japanese data is enough to achieve equal or better performance to XLSR on this dataset. Meng et al. (2022) study aspects of data quality when applying speech representation learning. They vary the *gender*, *content* and *prosody* of Librispeech for pre-training, and evaluate these models on the SUPERB benchmark from Yang et al. (2021). They found that pre-training performance does *not* decrease when the pre-training has a (significant) gender imbalance. Moreover, the complexity of the audio, i.e., whether or not the dataset contains audio with a large vocabulary of words, did not affect downstream performance either. However, they observe a significant decrease in downstream task performance when the pre-training audio is sped up, and a significant increase in performance when the pre-training audio is slowed down. Lam-Yee-Mui et al. (2023) continue pre-training XLSR with South-African soap opera broadcast data, which is shown to be an effective method to improve ASR performance for these low-resource languages, which are often used in a code-switching setting. Jacobs et al. (2023) build a dataset from radio broadcasts to evaluate hate speech detection in the low-resource Swahili and Wolof languages. Mateju et al. (2023) train a Swedish ASR system from scratch, using multiple labeled data sources, including television news from SVT, the Swedish public broadcasting organization. Lehečka et al. (2024) explore bi-lingual and tri-lingual pre-training on a multi-lingual oral archive dataset. Their pre-trained mono-lingual models performed well, but they did not have the computational resources to scale; they observed the best performance with the `large-v2` Whisper model from Radford et al. (2023).

6.2 Methodology

6.2.1 Pre-training and fine-tuning

In this chapter we use the contrastive learning approach of wav2vec 2.0 for pre-training speech representations. We also fine-tune the wav2vec 2.0 network for automatic speech recognition with labeled data using CTC loss (Graves et al., 2006). We do not make any modification to the network architecture, making use of the `BASE` (12 transformer layers) and `LARGE` (24 transformer layers) variant. We do not apply LayerDrop (Fan et al., 2020) to simplify the distributed data-parallel training. For fine-tuning and evaluating for English, we use Librispeech data. For fine-tuning and evaluating for Dutch, we use the Dutch split of multi-lingual Librispeech (Pratap et al., 2020) and the Dutch split of CommonVoice, version 18 (Ardila et al., 2020). We

normalize Dutch text such that the train and test data matches the letter vocabulary used in (Baevski et al., 2020a) for English, i.e., 26 letters and an apostrophe. All other punctuation, including hyphens, are substituted with spaces. Accented letters are converted to their closest form in the English alphabet.

6.2.2 Data quality simulation

To study the effect of data quality on self-supervised speech representation learning, we use the canonical 960 hour train set of the Librispeech (Panayotov et al., 2015) dataset as the baseline SSL training data. This training set contains ~281 k English utterances, with a minimum duration of 0.8 seconds, a maximum duration of 29.7 seconds, and an average duration of 12.3 seconds. In total, there are 2338 unique speakers (1210 male, 1128 female), with 51.7%/48.3% of the utterances being spoken by a male/female speaker. Each utterance is read speech, sourced from volunteers recording public domain audiobooks. This implies that there is little session variability for speakers, as it is unlikely volunteers would change recording conditions (microphone, room conditions, time factors such as sickness, or aging). We use the median length utterance in all chapters ("sessions") of each speaker for a validation set consisting of 2% (20 hours) of the train set. The `dev-clean` and `dev-other` set are used for choosing hyperparameters, and final (fine-tuning) results are reported on the `test-clean` and `test-other` set.

We augment the 960h train set of Librispeech in various ways to simulate different data qualities. For these augmentations, we make use of two other datasets. First, we use the MUSAN dataset (Snyder et al., 2015) as a source of background noise samples. MUSAN consists of three subsets (speech, music, noise). We only make use of the noise subset, which consists of 929 files with an average duration of 25 seconds, for a total of ~6 hours of data. There are three categories of noise:

1. technical sounds, e.g., dial tones
2. ambient sounds, such as creaking doors and police sirens
3. intelligible speech, i.e., recordings of crowd noises.

Secondly, the `FMA` (Free Music Archive) dataset (Defferrard et al., 2017) is used to provide music audio. We use the large subset, which consists of 104 k music segments which are all 30 second random chunks taken from the music recordings in the full database. The music has 16 top-level genres, including, e.g., classical, rock, and jazz.

Using these datasets, we simulate various forms of unclean data, which can arise when trying to pre-process a raw dataset into a pre-training dataset with the use of speech activity detection (SAD) tools.

Speaker overlap

One common scenario is a dialogue between two or more individuals, where the speaker turn is nearly instantaneous, making segmentation by SAD infeasible. In this case, one would use a high-quality diarization model to distinguish between speakers.

Another scenario is that, e.g., two speakers are speaking into the same channel at the same time. To disentangle the audio, one would require a speech separation model. However, for these simulations, we are simply interested in the effect on pre-training when these scenarios are not accounted for. To that end, we perform the following augmentations:

1. `1spk-concat` concatenates two random utterances from the *same session* of Librispeech; thus, all utterances contain only a single speaker. This acts as a baseline, where the audio is simply twice as long on average, with a minimal, but detectable, concatenation artifact, as the utterances are not in order.
2. `2spk-concat` concatenates two random utterances from *different speakers* of Librispeech. This simulates a speaker turn which was not detected by diarization tools. We do not control for the sex of the speakers, thus, there are same-sex and different-sex speaker turns. Compared to the baseline, the utterances, similarly, includes the concatenation artifact, but also adds the necessity for the SSL method to model two speakers.
3. `2spk-mix-x` mixes two utterances from two different speaker of Librispeech. This simulates the necessity of a speech separation model. At batch creation time, we randomly select a fraction of utterances in the batch to apply the mixing to. The remaining utterances are not modified, and thus clean data. The mixing is done with uniformly sampling a signal-to-noise ratio between 0 and 15 dB. We vary the fraction to 10 %, 33 %, or 50 %. Note that due variable lengths of utterances, these fractions are cannot be obtained precisely, but we aim to be as close as possible, always ensuring at least one utterance is selected.

Noise and music

The other scenarios we want to study is the inclusion of noise or music in utterances. In our archival dataset the presence of these are quite common. For example, a broadcast could simply be a recording of a concert, documentaries usually have background music, or interviews could be conducted in a busy street. To quantify the effect of these non-speech audio fragments on speech representation learning, we set-up the following augmentations:

1. `sub-noise-x` replaces one or more utterances in a batch with noise from MUSAN. Utterances are randomly cropped, or repeated, to match the exact length of the replaced utterance.
2. `sub-music-x` replaces one or more utterances in a batch with music from FMA. As all segments from `FMA` are 30 seconds long, we can, in each case, randomly crop to equal length of the Librispeech utterance.
3. `mix-noise-x` mixes one or more utterances in a batch with noise from MUSAN. We use all three categories of noise, including crowd noises.
4. `mix-music-x` mixes one or more utterances in a batch with music segments from `FMA`.

5. `mix-instr-x` mixes one or more utterances in a batch with music segments from `FMA`, which were stemmed to only contain the *instrumental* audio of the song track.
6. `mix-vocal-x` mixes one or more utterances in a batch with the music segments from `FMA`, which were stemmed to only contain the *vocal* audio of the song track. We applied a basic SAD on the vocal stem to filter out songs where were already instrumental before stemming.

For all settings, we select utterances to be mixed or replaced as done in `2spk-mix-x`, with fractions varying between 10 %, 33 %, and 50 %, ensuring at least one utterance is mixed or replaced. For mixing, we use a signal-to-noise ratio uniformly sampled between 0 and 15 dB. Stemming of `FMA` into vocal and instrumental tracks was done with Demucs[20] (Rouard et al., 2023), and SAD on potentially silent vocal tracks was done with Pyannote[21] (Bredin et al., 2020).

6.2.3 Archival data collection

To obtain a large quantity of Dutch audio data for self-supervised learning, we obtained television broadcasts within a 50-year period, from 1972 to 2022, collected by the Netherlands Institute for Sound and Vision (NIBG). This initial dump of data contained ~182 k hours of audio. First, we used meta-data to filter out genres which were unlikely to contain a good variety of speech, such as nightly news and sports broadcasting, and kept genres such as in-depth news analysis, quizzes, and documentaries. We also removed any broadcast which lasted longer than three hours, and de-duplicated some data by heuristically removing consecutive broadcasts with the same title, summary, and publication date. The remaining broadcasts contained ~ 81 k hours of audio, with most audio files having a length of 30 minutes to 1 hour. We refer to this 81 k hour dataset as `nibg` throughout this chapter. For pre-training, short segments are required, thus, we need the data to be segmented into short utterances.

6.2.4 Segmenting the broadcast data

To perform the segmentation, we initially attempted to apply SAD with Pyannote. However, we found the resulting segmentations to contain a lot of speaker overlap, music and noise. Moreover, initial pre-training experiments were unsuccessful, prompting experiments with simulating data quality, as detailed above. The second approach to segment the data involved Whisper (Radford et al., 2023) and WhisperX (Bain et al., 2023), with the idea that audio which can be successfully transcribed by Whisper contains speech with high-enough quality to be useful for self-supervised pre-training. Furthermore, WhisperX is able to perform speaker diarization on top of the transcription, potentially leading to utterances with a single speaker, like Librispeech. We experiment with 3 variations of segmenting with Whisper, and 5 variations segmenting with WhisperX.

[20] https://github.com/facebookresearch/demucs
[21] https://huggingface.co/pyannote/voice-activity-detection

First, we will first detail our methods using the `large-v2` version of Whisper (Radford et al., 2023). The Whisper model operates on 30 seconds of audio. Therefore, the ASR output intrinsically has a maximum length of 30 seconds. Whisper outputs a list of segments, with a *predicted* start and end-time for the spoken sentence. The first variation, `w-raw`, simply uses these output sequences, where utterances are simply cut by using the indicated start and end-time in each output segment. However, due to the nature of transcribing consecutive chunks of 30 seconds, many utterances of a single speaker are cut between multiple segments. These cuts are often indicated by ellipses (`...`) in the transcription. Moreover, the model often transcribes non-speech fragments by describing it, with phrases like `music` or `laughing`. Furthermore, the model tends to "hallucinate" on silent audio, an artifact from training on scraped, subtitled audio, which frequently has copyright information as a subtitle at the end, without paired speech. As a second and third variation, we apply post-processing to the Whisper output in an attempt to resolve these issues. The post-processing steps are as follows:

1. Filter out segments with transcriptions exactly matching (case-insensitive) these text phrases: `muziek`, `gelach`, `tv gelderland`, `applaus`, `gezang`, ***, and `.` (a single dot).
2. If the transcription of a segment does *not* end with sentence-ending punctuation (a dot, question or exclamation mark), we recursively merge it with the next segment until a (merged) segment does end with such punctuation. We do not merge segments if the gap between the current and next segment is longer than 3 seconds.
3. We remove segments which are shorter than a threshold, with the observations that many short segments are simply phrases like "yes" or "that is right"; long segments are more likely to be clean, varied speech.

If, due to the merging, segments end up longer than 30 seconds, we simply cut the segment in half recursively until all subsegments are shorter than 30 seconds. The two variations we use are `w-pp-1s` and `w-pp-3s`, which apply these post-processing steps on the segments from `w-raw`, with the threshold in the third step being respectively 1 and 3 seconds.

Secondly, WhisperX (Bain et al., 2023) operates in three stages. In the first stage, Pyannote SAD is applied to chunk the audio, with a cut and merge operation which aims to create chunks close to 30 seconds. These chunk can then be transcribed by Whisper (in our case, `large-v2`) in batches, allowing for an inference speed-up. Our first variation, `wx-asr`, uses the resulting output for segmenting the broadcast data. The second stage applies a forced alignment between the chunks and their respective transcriptions, using a wav2vec 2.0 model fine-tuned for ASR,[22] after which sentences are broken up in separate segments. The second variation, `wx-align`, uses the output of this stage as segmentations for the broadcast data. The third stage applies speaker

[22]For Dutch: https://huggingface.co/jonatasgrosman/wav2vec2-large-xlsr-53-dutch

diarization,[23] labelling each segment to a particular speaker. The third variant `wx-diar` uses the output after the third stage for segmentation. Our fourth and fifth variant apply, to `wx-diar`, the three post-processing steps mentioned above, similar to how `w-pp-1s` and `w-pp-3s` are created. However, instead of merging on punctuation in the second step, we merge on speaker labels instead, such that consecutive segments of the same speaker become a single segment. If this segment is longer than 30 seconds, we simply cut it, as described above. We use `wx-diar-1s` and `wx-diar-3s`, with a threshold for 1 and 3 seconds in the third post-processing step, respectively.

6.3 Experiments

6.3.1 Data quality

Setup

We pre-train the `BASE` wav2vec 2.0 variant on the 960 hours of training data from Librispeech, but apply various augmentations as described in Section 6.2.2. We use the default settings for pre-training as described in (Baevski et al., 2020a), i.e., diversity loss scaling of 0.1, L^2-penalty loss scaling of 10, gumbel-softmax temperature $\tau = 2$ to $\tau = 0.5$ with factor 0.999995, contrastive loss temperature $\tau_c = 0.1$, dropout of 10 % throughout the network, and the feature extractor CNN gradients are scaled by a factor 0.1. We also use 2 codebooks for quantization, with each codebook containing 320 entries of 128-dimensional codewords. However, we do not apply LayerDrop (Huang et al., 2016; Fan et al., 2020), as this complicates distributed data-parallel training (DDP).

For optimization, we use AdamW (Loshchilov and Hutter, 2019) with a weight decay of 0.1, $\beta_1 = 0.9$ and $\beta_2 = 0.98$. We use a batch size of 5 minutes for all settings, which means a batch contains up to 4.8 M frames, in order to fill up all available VRAM in a single NVIDIA A100 gpu. We train with a triangular cyclic learning rate schedule, with 25 k linear steps up and down. The minimum learning rate is a 100 times smaller than the maximum learning rate. For each condition we scan for the best maximum learning rate (LR) in the cycle with a 2-phased approach. In the first phase we attempt LR $3.2 \cdot 10^{-5}$, 10^{-4} and $3.2 \cdot 10^{-4}$ with 50 k training steps, i.e., a single cycle. In the second phase we train for 400 k steps, i.e., 8 cycles, with the best learning rate in phase 1, and 2 additional LRs, based a reasonable step size upwards or downwards, respectively:

1. If $3.2 \cdot 10^{-5}$ was best in phase 1, we also attempt LR $1.0 \cdot 10^{-5}$ and $1.8 \cdot 10^{-5}$.
2. If 10^{-4} was best in phase 1, we also attempt LR $5.6 \cdot 10^{-5}$ and $1.8 \cdot 10^{-4}$
3. If $3.2 \cdot 10^{-4}$ was best in phase 1, we also attempt LR $5.6 \cdot 10^{-4}$ and $1.0 \cdot 10^{-3}$

The best LR is chosen according to the run with the lowest validation loss. The validation split, made beforehand, holds out a single utterance from each session in

[23] https://huggingface.co/pyannote/speaker-diarization

Table 13: *The WER on Librispeech after fine-tuning pre-trained models, with simulated variations in pre-training data quality. Fine-tuning results are shown for fine-tuning with 10 minutes of labels and 100 hours of labels. We also show the minimum self-supervised validation loss for each pre-training condition, where* DIV *indicates that the training diverged and no pre-trained model was obtained. This was the case for* `mix-music` *and* `mix-vocal`*, for all three fractions.*

		10 m labels		100 h labels	
pre-training data	SSL loss	`test-clean`	`test-other`	`test-clean`	`test-other`
`baseline`	11862	59.56%	69.90%	12.40%	31.48%
`1spk-concat`	11775	62.01%	72.58%	12.82%	32.92%
`2spk-concat`	12390	75.56%	84.16%	16.84%	39.24%
`2spk-mix-10`	12021	62.51%	72.30%	12.23%	31.94%
`2spk-mix-33`	12264	63.86%	73.63%	13.36%	33.39%
`2spk-mix-50`	12527	71.74%	80.51%	14.65%	36.28%
`sub-noise-10`	12357	63.86%	74.61%	13.30%	33.91%
`sub-noise-33`	13518	77.74%	86.77%	18.50%	41.99%
`sub-noise-50`	14605	87.02%	92.19%	23.92%	48.40%
`sub-music-10`	12321	66.45%	76.56%	14.39%	35.31%
`sub-music-33`	12832	69.88%	79.30%	15.36%	37.45%
`sub-music-50`	13383	75.33%	84.97%	17.41%	40.50%
`mix-noise-10`	12042	61.56%	71.72%	12.65%	32.30%
`mix-noise-33`	12395	65.71%	75.67%	13.25%	33.24%
`mix-noise-55`	12640	68.73%	78.19%	14.81%	35.88%
`mix-music->=10`	DIV	/	/	/	/
`mix-vocal->=10`	DIV	/	/	/	/
`mix-instr-10`	12055	64.00%	73.24%	12.53%	31.80%
`mix-instr-33`	12463	69.60%	78.42%	14.57%	35.56%
`mix-instr-50`	12887	70.76%	79.55%	15.60%	37.41%

the training data of Librispeech. We do *not* apply the data quality simulation to the validation data, as we want to evaluate the quality of the speech representations on data which is similar to the fine-tuning dataset.

Fine-tuning is done with the 10 min and 100 hour labeled data conditions, using Librispeech training data, as done in (Baevski et al., 2020a). We use the default tri-stage learning rate with a 10 % warm up, 40 % constant, and 50 % exponential decay

phase, with the Adam optimizer (Kingma and Ba, 2015), with $\beta_1 = 0.9$, $\beta_2 = 0.98$, and no weight decay. For both settings we use an initial LR of $5 \cdot 10^{-7}$, a constant LR of $5 \cdot 10^{-5}$, and a final LR of $2.5 \cdot 10^{-6}$. We mask 5 % of the latent speech feature sequence, and use a dropout of 10 %. Inference is done greedily with letter-decoding, we do not use a language model. For 10 minutes of labels we train for 12 k steps, for 100 hours of labels we fine-tune for 80 k steps. The feature CNN is frozen for the first 5 k steps.

Results

We show fine-tuning performance of all data quality simulation in Table 13. First, we see that `baseline`, where we trained on the vanilla Librispeech training dataset without any data manipulations, has the best performance, compared to all other settings. There is one exception, namely for `2spk-mix-10` the 12.23% on `test-clean` with 100 h labeled data is slightly better than the baseline of 12.40 %. Secondly, we see that there is a minor performance drop when concatenating two utterances from the same speaker (`1spk-concat`), and a more significant performance drop when utterances from two different speakers are concatenated (`2spk-concat`). Thirdly, we see that substituting 10% of the data in a batch with pure noise (`sub-noise-10`) has a minor effect on performance, comparatively with concatenating the same speaker to all utterances (`1spk-concat`). A much larger effect is visible when the fractions of substitutions is increased to 33% and 50%. We also observe that noise substitutions are less harmful than music substitutions in minor fractions (`sub-music-10`), but for the larger fractions, music substitutions lead to less degradation, although the performance is still substantially worse compared to `baseline`. Fourthly, we observe that for all three conditions of `mix-music` and `mix-vocal`, pre-training is not stable. We only managed to find a converging learning rate when mixing instrumental music. For `mix-music` (which can be any combination of vocal and instrumental music), or `mix-vocal`, we see that even when mixing a small amount of the data in a batch (10 %) the contrastive loss diverges. Fifthly, we see mixing in a second speaker (`2spk-mix`) only leads to relatively small performance drops. The performance of `2spk-mix` sits between the performance of the baseline and `2spk-concat`. Note that the impact of mixing a speaker in 50 % of the data is on par with substituting 10 % of the data with music.

6.3.2 Effective pre-processing methods

From the analysis of data quality above, it seems paramount that pre-training data contains as little music as possible. To this end, we proposed the pre-processing methods described in Section 6.3.2, with the hope that music sections will either be ignored, or transcribed as such, by the Whisper model. As pre-processing the whole `nibg` dataset with Whisper is computationally intensive, we decided to first analyze the performance of different variations on a subset of `nibg`. At random, we selected 4000 broadcasts, with a total duration of ~2500 hours of data. We applied the various

Table 14: *The result of applying 3 pre-processing techniques (naively, using Whisper, and using WhisperX) on a subset of 4000 files of* `nibg`*. The "discarded" column indicates the amount of data which was left unused after the pre-processing. Note that* `wx-diar-1s` *and* `wx-diar-3s` *are both a cleaned version of* `wx-diar`*, respectively filtering out segments shorter than 1 or 3 seconds.*

subset	segments	duration	discarded	avg	min
naive					
sequential (`seq`)	296,357	2469 h	0.65%	30.0	30.0
Whisper (Radford et al., 2023)					
raw output (`w-raw`)	2 183 647	1 998 h	19.63%	3.29	0.5
clean 1 s (`w-pp-1s`)	2 089 314	1 868 h	24.86%	3.22	1.0
clean 3 s (`w-pp-3s`)	1 156 881	1 370 h	44.90%	4.26	3.0
WhisperX (Bain et al., 2023)					
transcribed (`wx-asr`)	347 178	2 158 h	13.20%	22.37	0.5
aligned (`wx-align`)	1 680 228	1 779 h	28.44%	3.81	0.5
diarized (`wx-diar`)	1 674 252	1 773 h	28.65%	3.81	0.5
clean 1 s (`wx-diar-1s`)	588 866	1 725 h	30.59%	10.55	1.0
clean 3 s (`wx-diar-3s`)	473 831	1 663 h	33.10%	12.63	3.0

Table 15: *For each pre-processing subset shown in Table 14 we show the lowest SSL validation loss (batch size of 5 minutes), the step at which this loss was reached, and fine-tuning performance on two datasets, the Dutch multi-lingual Librispeech and CommonVoice. Evaluation is done on both test set of the respective datasets, shown as WER in %. The in-domain test set is italicized. The * indicates that training diverged.*

	SSL		ft. on `MLS-train`		ft. on `CV-train`	
subset	loss	step	`MLS-test`	`CV-test`	`MLS-test`	`CV-test`
`seq`	15255	400 k	*31.0*	58.0	53.2	*39.7*
`w-raw`	24765*	270 k	*100*	100	100	*100*
`w-pp-1s`	25066*	15 k	*100*	100	100	*100*
`w-pp-3s`	15267	400 k	*25.3*	49.3	46.7	*31.3*
`wx-asr`	15061	355 k	*31.3*	57.7	53.1	*39.8*
`wx-align`	15281	400 k	*27.6*	52.6	48.1	*33.9*
`wx-diar`	15203	400 k	*27.6*	51.3	48.1	*33.8*
`wx-diar-1s`	15076	400 k	*27.2*	52.4	47.5	*34.6*
`wx-diar-3s`	15009	400 k	*27.9*	53.0	49.3	*35.4*

pre-processing techniques to this subset. The resulting subsets and their statistics are displayed in Table 14. The `sequential` subset can be seen as a baseline with minimal pre-processing; the discard of 0.65% of data is simply the truncation at the end of each file, as we discard the last segment of each file if it is not exactly 30 seconds. Furthermore, we note that `wx-diar-3s` discards a lot of data, almost a third, but has the closest characteristics to Librispeech, with a minimum duration of 3 seconds for each utterance, and an average duration of 12.63 seconds.

For each subset we perform self-supervised pre-training with hyperparameters equivalent to Section 6.3.1. First, we pre-train with all subsets, using a batch size of 5 minutes. On the most promising subsets we also pre-train with a batch size of 40 minutes. Instead of a learning rate scan, we simply use the cyclic schedule, with a minimum LR of 10^{-6} and a maximum LR of 10^{-4}. We selected the checkpoint with the lowest validation loss for fine-tuning. As validation data we combine the validation split of Dutch MLS (MLS) and Dutch CV.

The checkpoints are fine-tuned and evaluated with the respective splits of the Dutch CV and MLS datasets. We note the training splits as `MLS-nl` and `CV-nl`, and the test sets as `MLS-test` and `CV-test`. Fine-tuning is done with the exact same configuration as the 100 h fine-tuning condition in Section 6.3.1, including the maximum learning rate of $5 \cdot 10^{-5}$.

The fine-tuning results with a pre-training batch size of 5 minutes are shown in Table 15. Firstly, looking at Whisper-based segmentation, we see that pre-training with `w-raw` data leads to divergence. This is also the case for `w-pp-1s`. However, `w-pp-3s` has the best overall fine-tuning performance. Secondly, looking at WhisperX-based segmentation, we see that the first stage, `wx-asr`, together with the naive baseline `seq`, have the worst fine-tuning performance (ignoring divergence). It is noticeable that `seq` and `wx-asr` have very similar performance, even though for `wx-asr` 13.2 % of the data is discarded, while `seq` only discards a marginal 0.65 %. Doing a forced alignment with a pre-trained wav2vec 2.0 model, as done with `wx-align`, leads to a large improvement in performance. Performing diarization (as done with `wx-diar`) seems to barely affect the performance. Finally, we see that the cleaning steps applied on `wx-diax` only decrease performance.

When pre-training with a batch size of 40 minutes (not shown in Table 15), we observed a different ranking for `wx-diar` and `wx-diar-3s`. The `w-pp-3s` dataset still had the best performance, with 16.5 % WER when fine-tuning and evaluating on MLS. However, for `wx-diar-3s`, `wx-diar`, and `seq` we respectively observed a WER of 17.1%, 17.5%, and 19.7%. When fine-tuning and evaluating on CV, we observed, respectively for `w-pp-3s`, `wx-diar-3s`, `wx-diar` and `seq`, a WER of 12.8%, 14.0%, 14.6%, and 17.7%. We note that these performances all substantially improved compared to pre-training with a batch size of 5 minutes, indicating that all pre-processing techniques are suitable for scaling-up, although we do not know to what extent. Finally, we note that

Table 16: *The WER on the test splits of Dutch multi-lingual Librispeech and CommonVoice, after fine-tuning various pre-trained models, including those from previous work (Baevski et al., 2020a; Conneau et al., 2021). For both fine-tuning conditions we evaluate on the in-domain test set (in italics) and the out-of-domain test set. Note that* `MLS` *and* `CV` *are multi-lingual, while* `MLS-nl` *and* `CV-nl` *are the respective Dutch subsets. We shortened* `MLS-test` *and* `CV-test` *to* `MLS-t` *and* `CV-t`*.* `VP-nl` *is the Dutch subset of the VoxPopuli datset. (Wang et al., 2021).*

	SSL configuration			ft. on `MLS-nl`		ft. on `CV-nl`	
model	data	batch size	steps	`MLS-t`	`CV-t`	`MLS-t`	`CV-t`
pre-trained models by (Baevski et al., 2020a) and (Conneau et al., 2021)							
`LARGE`	`LS`	80 min	400k	*15.9*	34.9	33.4	*18.5*
`LARGE`	`VP-nl`	80 min	250k	*16.1*	23.9	25.6	*12.8*
`LARGE`	`MLS,CV,BABEL`	80 min	250k	*13.0*	*25.1*	*20.5*	*12.0*
baseline pre-training on English and Dutch datasets							
`BASE`	`LS`	5 min	400k	*25.8*	53.1	45.4	*32.8*
`BASE`	`CV-nl`	5 min	400k	*38.9*	67.3	70.5	*56.3*
`BASE`	`MLS-nl`	5 min	400k	*31.3*	62.7	50.5	*41.2*
pre-training with 55 k hours of Dutch data							
`BASE`	`nibg-pp`	5 min	400k	*30.9*	55.3	45.5	*32.1*
`BASE`	`nibg-pp`	40 min	400k	*16.8*	27.8	26.2	*13.8*
`LARGE`	`nibg-pp`	40 min	500k	*13.3*	21.2	22.5	*9.9*

the run-time for creating the `wx-diar-3s` dataset is lower than `w-pp-3s`, respectively 40 and 50 hours, using a single NVIDIA A100 GPU.

6.3.3 SSL with 55 k hours of Dutch audio data

Setup

In the last set of experiments we scale the pre-training to the whole `nibg` dataset. To save time and resources, we apply the `wx-diar-3s` pre-processing technique, although based on Section 6.3.2 we expect that the `w-pp-3s` method should have lead to slightly better performance. After applying the `wx-diar-3s` pre-processing on the raw 81 k hour dataset, we obtain the pre-training dataset `nibg-pp` with 55 671 hours of data, having discarded 31 % of the raw dataset (~25 k hours). The dataset has 16 876 597 segments, with an average of 11.9 seconds, and a minimum and maximum length of respectively 3 and 30 seconds. The length distribution peaks at 3 seconds, with ~1.8 M utterances with a length between 3.0 and 3.1 seconds, compared to ~75 k

utterances with a length between 11.9 and 12 seconds, and ~90 k utterances with a length between 29 and 30 seconds.

We pre-train with the BASE and LARGE variant of wav2vec 2.0. We use the same configuration as detailed in Section 6.3.1. However, we change the learning rate schedule to a two-stage approach as in (Conneau et al., 2021) for a fairer comparison. The two-stage schedule linearly warms-up for the first 10% of total steps, the remaining 90% of steps decay the LR with cosine annealing. The initial LR is 1000 times smaller than the peak LR, while the final LR is 100 times smaller. For both network variants, we first perform a grid scan for the (peak) LR with a batch size of 5 minutes. The grid has a range of $[10^{-4}, 10^{-3}]$ and a step size of 10^{-4}. We use the best-performing LR to scale to a batch of 40 minutes with the square root scaling law (Malladi et al., 2022). For a baseline, we also pre-train with English Librispeech (960 hours), Dutch multi-lingual Librispeech (1550 hours), and Dutch CommonVoice data (50 hours). For the baseline we use a 5 minute batch size and use the BASE model variant. Fine-tuning and evaluation is done on multi-lingual Librispeech and CommonVoice as in Section 6.3.2, without scanning over any hyperparameters; only the initial SSL checkpoint is varied.

Results

The results are shown in Table 16. First, we fine-tune pre-existing checkpoints, namely the LARGE network pre-trained with many GPUs on Librispeech[24] in (Baevski et al., 2020a), the Dutch split of VoxPopuli[25] (Wang et al., 2021) in (Conneau et al., 2021), and the multi-lingual pre-training[26] on the MLS, CV and BABEL, also in (Conneau et al., 2021). We see that the Librispeech pre-training has a better WER on MLS compared to the network pre-trained on VoxPopuli data, but only in the in-domain setting. For CommonVoice, the multi-lingual pre-training has the best performance. When looking at our baseline experiments with a small batch size, we see that pre-training on Librispeech, compared to Dutch multi-lingual Librispeech, or Dutch CommonVoice, is more effective, with substantially lower WERs on both multi-lingual Librispeech and CommonVoice evaluation data. We think this is quite notable, as the Librispeech pre-training is *out-of-domain* both linguistically (English versus Dutch) and acoustically (in the case of CommonVoice). When we pre-train on 55 k hours of nibg-pp with the BASE network and the 5 minute batch size, we still see that the Librispeech pre-training outperforms on MLS evaluation data, although for the CV data we see slightly better performance with nibg-pp. When we scale the batch size from 5 to 40 minutes, we see the WERs reduce more than 50%, closely matching the performance in the pre-existing pre-trainings (Conneau et al., 2021). When we scale the network from BASE to LARGE, we see that the WERs are mostly lower compared to the multi-lingual training by Conneau et al. (2021). Only on the in-domain evaluation on MLS does the multi-lingual pre-training slightly outperform our mono-lingual pre-

[24]https://huggingface.co/facebook/wav2vec2-large

[25]https://huggingface.co/facebook/wav2vec2-large-nl-voxpopuli

[26]https://huggingface.co/facebook/wav2vec2-large-xlsr-53

training. Note that the multi-lingual pre-training is only evaluated on in-domain data, while our `nibg-pp` pre-training is only evaluated on out-of-domain data.

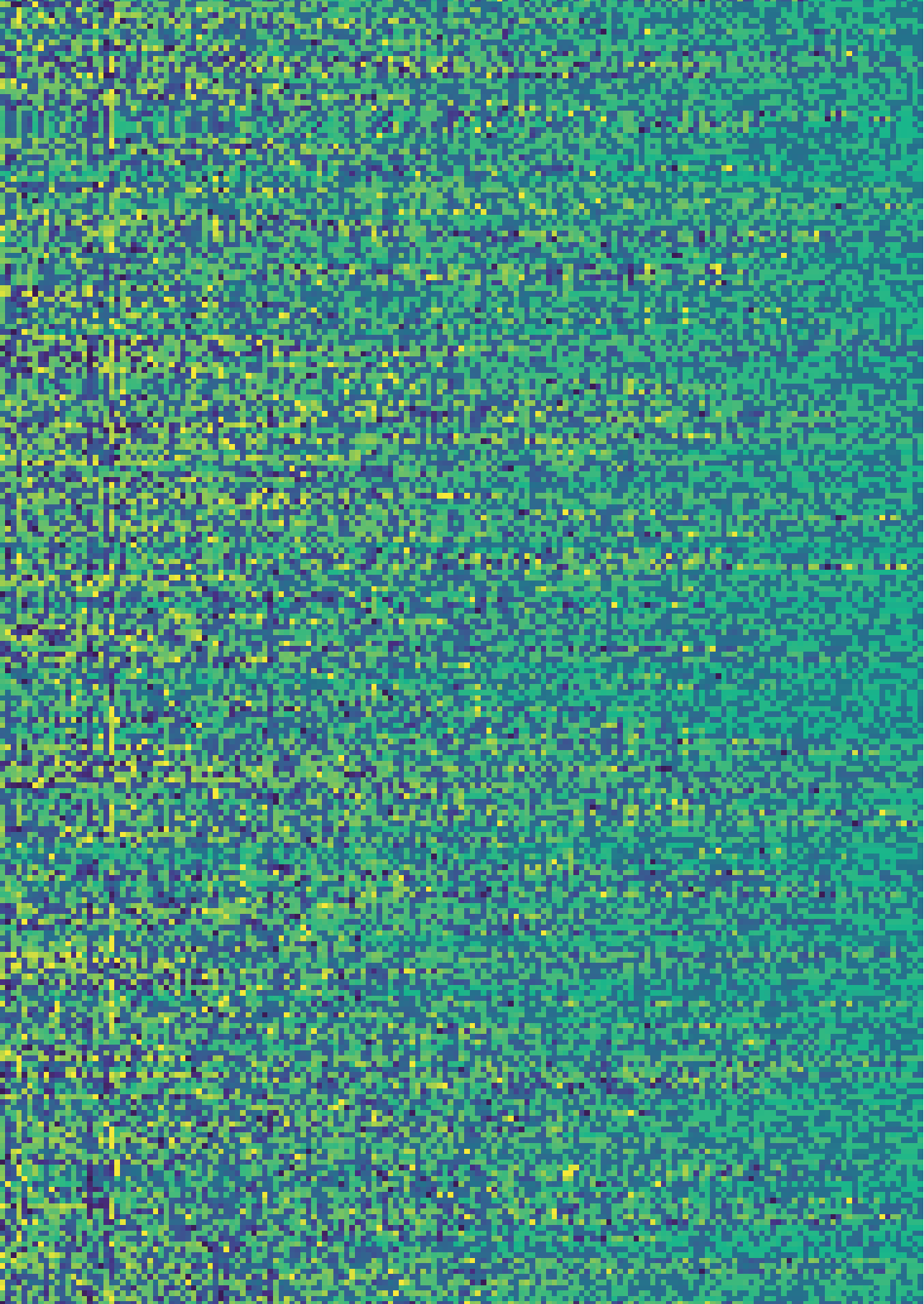
6.4 Discussion and conclusions

Our first research question focused on the data requirements for self-supervised learning of speech representations with the contrastive wav2vec 2.0 framework. Most notably, we observed that mixing music with the speech utterance lead to divergence during training. Only the music consisting solely of instrumental tracks had converging behavior, most likely because instrumental music can be seen, acoustically, as noise instead of speech. Overall, each studied data quality condition lead to worse fine-tuning performance, indicating that wav2vec 2.0 requires datasets as clean as Librispeech to perform optimally. We expect that other SSL techniques, like HuBERT (Hsu et al., 2021) and DinoSR (Liu et al., 2023), are similarly overfitted, on a meta-level, to Librispeech. In particular, WavLM (Chen et al., 2022a) *requires* utterances to be single-speaker, as the pre-training relies on mixing a second speaker into the utterance, which the models needs to learn to separate. We hope that future work on speech SSL is more explicit in the data quality assumptions, and can focus on more robust methods which work with noisier data.

Our second research questions involved the required steps to create a qualitative pre-training datasets. This question is quite specific to our `nibg` dataset, but we think our observations generalize for all cases with noisy (raw) datasets. We found naive speech activity detection to be insufficient on our television broadcasting data, due to the relatively high error rate regarding music, and overlapped speech. We found Whisper to be an effective, albeit computationally intensive, tool for pre-processing the audio. Noticeably, we needed to filter out segments shorter than 3 seconds to have a robust pre-training. Coincidentally, Librispeech utterances are at least 3 seconds, however, we did manage to have successful pre-trainings with the WhisperX-based subsets which had a minimum length of 0.5 seconds. We assume that removing short segments was an effective heuristic to remove low-quality speech utterances, due to limited acoustic signals, or noisy overlapped speech. We note that our naive sequential baseline, which simply segmented by taking consecutive 30 seconds chunks of the audio, did successfully pre-train, albeit with lower performance. We are curious whether, at the limit of computational resources, there is a convergence between the naive baseline, and the "smart" segmentation. We hope that future work can analyze whether simply training for more epochs closes the performance gap between clean and noisy data, or whether clean data inherently leads to better learned representations. For now, we conclude that using clean speech data is more efficient.

Our last research questions involved a comparison between mono-lingual and multi-lingual pre-training. We found that our mono-lingual pre-training had better performance on the CommonVoice dataset, but equivalent performance on the multi-lingual Librispeech dataset. We were also surprised by the relatively good performance of

fine-tuning for Dutch when using a model pre-trained only using (English) Librispeech data. This could be seen for the LARGE model by (Conneau et al., 2021), but also our own pre-trained BASE model. The good performance of English Librispeech could be due to the fact that it is very clean data, and that SSL techniques, including wav2vec 2.0, are initially developed using Librispeech data. We conclude that monolingual pre-training seems more robust to out-of-domain data *within* the same language, although we believe future work could analyze the distinction between in-domain data during pre-training, and during fine-tuning.



7 Conclusions

In which our adventurer reflects on the journey through speech representation research.

7.1 Reflections on presented work

The research presented in this dissertation has covered two aspects of self-supervised speech representation learning. First, in Chapter 2, Chapter 3 and Chapter 4, we studied the adaptability of speech representations learned with wav2vec 2.0, by fine-tuning the network in various settings. Later, in Chapter 5 and Chapter 6, we studied the self-supervised learning process of wav2vec 2.0 in more detail, with an analysis on the batch size, data quality assumptions, and a mono-lingual versus multi-lingual dataset comparison.

In Chapter 2, we were one of the first to show that wav2vec 2.0 could be fine-tuned for speaker recognition. However, our work considered extracting speaker embeddings directly from the last transformer layer, with various pooling techniques. Instead of pooling the transformer output directly, it became standard to add a relatively large network, such as X-vector or ECAPA-TDNN, on top of the transformer layers (Chen et al., 2022a), while also using a weighted sum of all encoder layers. While these additions achieved state-of-the-art speaker recognition performance, we believe large task-specific heads will not unify speech technology tasks.

In Chapter 3, we showed how wav2vec 2.0 enables speaker recognition in low-resource settings. Our work considered relatively small datasets, namely 100 hours of audio with roughly 50 k utterances. We observed that the (small) dataset with the maximum number of speakers and session had the best performance. Note that the reduced versions of VoxCeleb2 were *smaller* than using a single 3 second segment from each session of each speaker in VoxCeleb2, which results in 113 hours of data compared to the full dataset with 2314 hours. For ASR, it was seen that wav2vec 2.0 could be fine-tuned with 10 minutes of audio. However, the result with 10 minutes of data uses an impractically large beam search with a n-gram language model. Reflecting on our work, it would have been interesting to test smaller datasets than 100 hours, although for speaker recognition we do not expect any reasonable results with only 10 minutes of data, as there is no beam search equivalence.

We also note that the pre-training of wav2vec 2.0 is primarily designed for ASR. Specifically, the contrastive loss only considers negative samples within the same

utterance, and thus, from the same speaker and session. The loss function of WavLM is designed to model speaker information by learning to separate a second speaker which is mixed into pre-training data. Separating the second speaker is a noise reduction task, but to perform this separation, the model needs to be able to recognize which parts of the speech signal are from a background speaker, which should require some form of modelling speaker information. Therefore, it would be interesting to repeat the experiments in Chapter 3 with the WavLM model.

In Chapter 4, we analyzed the feasibility of a MTL model for speech and speaker recognition. We observed that the orthogonality of speech and speaker made performance a trade-off, these tasks did not jointly benefit from each other. It could be that the `BASE` wav2vec 2.0 network did not have a large enough capacity to model both tasks, and that experiments with the `LARGE` network would show better performance. However, it could also be that, like mentioned above, the wav2vec 2.0 representations do not have enough intrinsic speaker information, and that fine-tuning speaker recognition is too disruptive to the learned representation. Here we are also interested in repeating the experiments with WavLM. If WavLM alleviates the performance trade-off, even if only moderately, this hints that research towards more unified self-supervised learning techniques is a promising direction. We are also curious whether a better dataset could prevent the MTL model from failing to generalize to out-of-domain data.

In Chapter 5, we found a direct relationship between the amount of data seen during self-supervision, and downstream task performance, independent from the SSL batch size, for wav2vec 2.0. Due to the fact that seminal SSL papers, originating from industry research labs, reported results with only large batch sizes, we thought applying these techniques with small batch sizes might be infeasible. From our results, we know small batches do converge, where performance improves as the product of batch size and the number of iterations increases. Due to this relationship, we think there are possibilities for practitioners with low(er) computational budgets to contribute to SSL methodology research. As long as the amount of data seen is kept constant, comparisons can be made regarding, e.g., the data efficiency of SSL methods.

In Chapter 6, we created a large mono-lingual dataset from television broadcast data. We learned that wav2vec 2.0 requires specific data quality conditions, where pre-training needs to be done on clean, prepared speech. Specifically, we observed that the presence of music led to divergence. We think it is plausible that these observations generalize to other methods like HuBERT, WavLM and DinoSR. Thus, while it is often argued that self-supervised learning is beneficial due to the ease of scaling an unlabeled dataset, in practice these data assumptions enforce soft label requirements on the data. Moreover, we saw that mono-lingual pre-training was able to achieve better performance than multi-lingual pre-training with the same computational budget, but this was dependent on the dataset. It would be valuable to perform an evaluation on a dataset that is out-of-distribution for both the mono-lingual and

multi-lingual model, as the multi-lingual model was pre-trained on the training set of the evaluation datasets, while the mono-lingual model was not.

7.2 Future developments

How will we achieve a *single* artificial neural network for all speech technology tasks? While foundation models like wav2vec 2.0, HuBERT, and WavLM, can be fine-tuned, separately, for all tasks in the SUPERB benchmark, there is no multi-task model capable of all tasks simultaneously. In Chapter 4 we saw that building such a multi-task learning (MTL) model negatively impacts performance, and more research is required to understand how we can combine these orthogonal tasks. The Whisper model is a promising direction towards a unified MTL model for speech technology tasks, capable of ASR, speech translation, and some form of speech activity detection. However, Whisper does not consider speaker, prosodical or generational aspects of speech technology. What are our projections for future developments towards a single MTL model?

Richer datasets

To enable the fine-tuning of a MTL model for speech and speaker recognition, we believe we need a dataset with transcripts and high session variability. For the speaker diarization task, it could also be valuable for such a dataset to include speaker turns. We speculate that synthetic data generation, with high-quality text-to-speech models, could be an approach to generate better fine-tuning datasets, although currently, model collapse is an open problem.

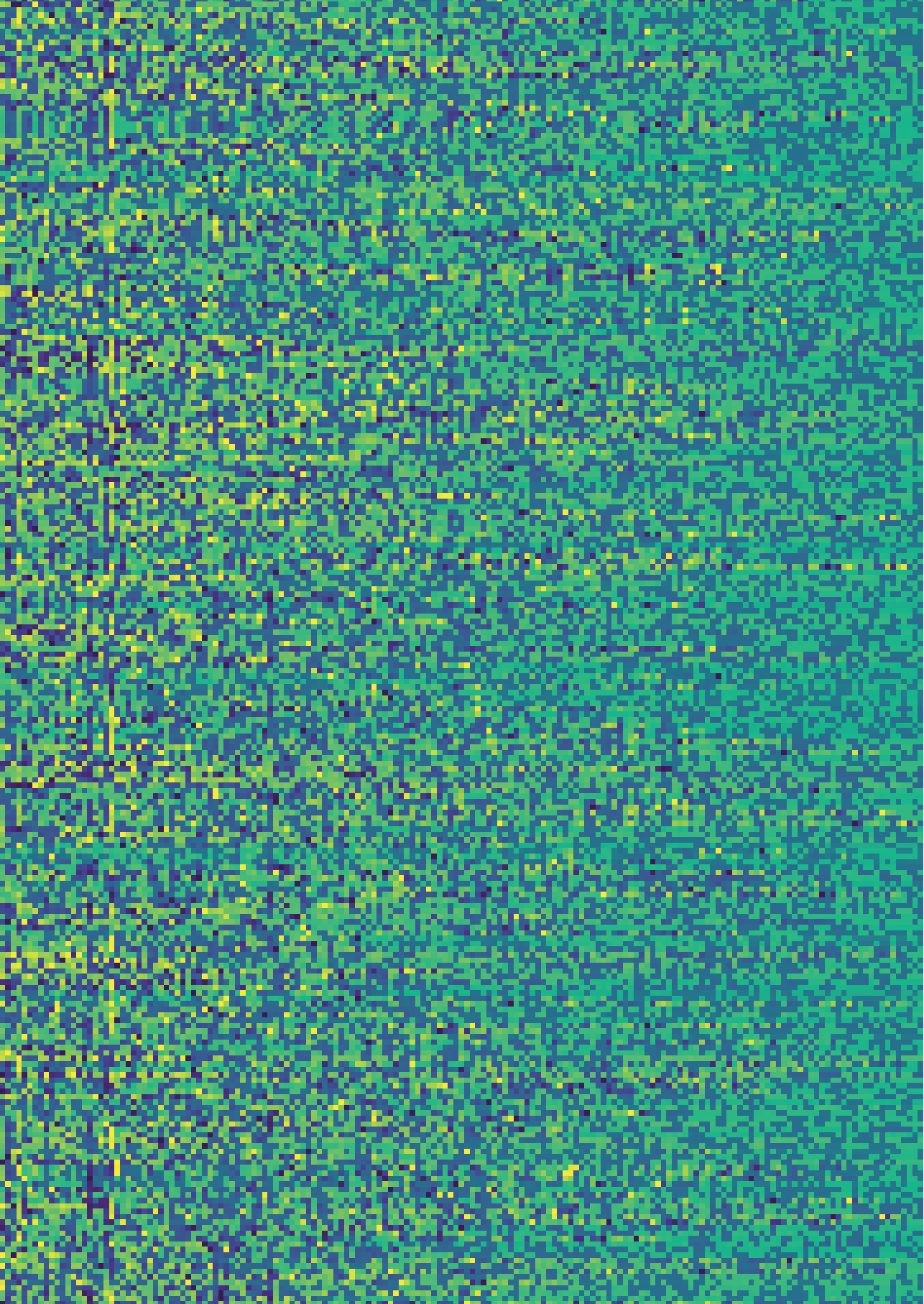
Tokenization of speech tasks

A rich dataset as described above, could allow models like Whisper to be fine-tuned to include non-text tokens, related to e.g., speaker labels, speaker turns and emotion. Thus, we envision that other speech technology tasks can be achieved by having a richer transcription output. This trend of directly learning tasks in an end-to-end fashion has been shown to be effective throughout development of deep learning methods, and we believe that future work in speech technology will also focus on end-to-end approaches, which include more speech technology tasks directly into encoder-decoder transformer models.

Improving SSL methods

Currently, most SSL techniques are primarily designed for, and evaluated on, the ASR downstream task. WavLM is a noticeable exception, with a specific goal of modelling speaker information, and it has the best overall performance on the SUPERB benchmark. We believe that unifying speech SSL methods, so that not only phonetic information needs to be learned, is a promising future direction. Moreover, it would be valuable to develop SSL methods which are more robust to noise and music, so

that it is easier to scale the dataset size. Furthermore, it seems plausible that scaling the data and model parameters might eventually hit limits in performance gains. We believe that more research effort could be spent on increasing the data efficiency of SSL methods, e.g., a benchmark could be created where SSL methods have a fixed amount of data seen during pre-training.



Bibliography

Adi, Y., Zeghidour, N., Collobert, R., Usunier, N., Liptchinsky, V., and Synnaeve, G. (2019). To reverse the gradient or not: An empirical comparison of adversarial and multi-task learning in speech recognition., in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 3742–3746.

Alam, J., Bhattacharya, G., and Kenny, P. (2018). Speaker Verification in Mismatched Conditions with Frustratingly Easy Domain Adaptation., in *Odyssey 2018 The Speaker and Language Recognition Workshop*, 176–180.

Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., et al. (2016). Deep Speech 2: End-to-End Speech Recognition in English and Mandarin., in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, (New York, NY, USA: JMLR.org), 173–182.

Ardila, R., Branson, M., Davis, K., Kohler, M., Meyer, J., Henretty, M., et al. (2020). Common Voice: A Massively-Multilingual Speech Corpus., in *Proceedings of the Twelfth Language Resources and Evaluation Conference*, (Marseille, France: European Language Resources Association), 4218–4222. Available at: https://aclanthology.org/2020.lrec-1.520

Ashihara, T., Moriya, T., Matsuura, K., and Tanaka, T. (2023). Exploration of Language Dependency for Japanese Self-Supervised Speech Representation Models., in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1–5. doi: 10.1109/ICASSP49357.2023.10096318

Baevski, A., Hsu, W.-N., Xu, Q., Babu, A., Gu, J., and Auli, M. (2022). Data2vec: A general framework for self-supervised learning in speech, vision and language., in *International Conference on Machine Learning*, 1298–1312. Available at: https://proceedings.mlr.press/v162/baevski22a.html

Baevski, A., Schneider, S., and Auli, M. (2020b). vq-wav2vec: Self-Supervised Learning of Discrete Speech Representations., in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, (OpenReview.net). Available at: https://openreview.net/forum?id=rylwJxrYDS

Baevski, A., Zhou, Y., Mohamed, A., and Auli, M. (2020a). wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations., in *Advances in Neural Information Processing Systems*, (Curran Associates, Inc.), 12449–12460. Available at: https://proceedings.neurips.cc/paper/2020/hash/92d1e1eb1cd6f9fba3227870bb6d7f07-Abstract.html

Bain, M., Huh, J., Han, T., and Zisserman, A. (2023). WhisperX: Time-Accurate Speech Transcription of Long-Form Audio., in *INTERSPEECH 2023*, 4489–4493. doi: 10.21437/Interspeech.2023-78

BenZeghiba, M. F., and Bourlard, H. (2003). On the combination of speech and speaker recognition., in *Proc. 8th European Conference on Speech Communication and Technology (Eurospeech 2003)*, 1361–1364. doi: 10.21437/Eurospeech.2003-421

Bredin, H., Yin, R., Coria, J. M., Gelly, G., Korshunov, P., Lavechin, M., et al. (2020). Pyannote.Audio: Neural Building Blocks for Speaker Diarization., in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 7124–7128. doi: 10.1109/ICASSP40776.2020.9052974

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems* 33, 1877–1901.

Busso, C., Bulut, M., Lee, C.-C., Kazemzadeh, A., Mower, E., Kim, S., et al. (2008). IEMOCAP: Interactive emotional dyadic motion capture database. *Language resources and evaluation* 42, 335–359. Available at: https://sail.usc.edu/publications/files/bussolre2008.pdf

Bălan, D. A., Ordelman, R. J., Truong, K., and Heuvel, H. van den (2024). Benchmarking and Research Infrastructures: Evaluating Dutch Automatic Speech Recognition., in *CLARIAH Annual Conference 2024*.

Cai, D., Wang, W., and Li, M. (2021). An Iterative Framework for Self-Supervised Deep Speaker Representation Learning., in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 6728–6732. doi: 10.1109/ICASSP39728.2021.9414713

Cao, Y.-H., and Wu, J. (2021). Rethinking Self-supervised Learning: Small is Beautiful. *arXiv preprint arXiv:2103.13559*. Available at: https://arxiv.org/abs/2103.13559

Chen, C., Zhang, J., Xu, Y., Chen, L., Duan, J., Chen, Y., et al. (2022b). Why do We Need Large Batchsizes in Contrastive Learning? A Gradient-Bias Perspective., in *Advances in Neural Information Processing Systems*, (Curran Associates, Inc.), 33860–33875. Available at: https://proceedings.neurips.cc/paper_files/paper/2022/file/db174d373133dcc6bf83bc98e4b681f8-Paper-Conference.pdf

Chen, G., Chai, S., Wang, G.-B., Du, J., Zhang, W.-Q., Weng, C., et al. (2021). GigaSpeech: An Evolving, Multi-Domain ASR Corpus with 10,000 Hours of Transcribed Audio., in *Proc. Interspeech 2021*, 3670–3674. doi: 10.21437/Interspeech.2021-1965

Chen, S., Wang, C., Chen, Z., Wu, Y., Liu, S., Chen, Z., et al. (2022a). WavLM: Large-Scale Self-Supervised Pre-Training for Full Stack Speech Processing. *IEEE*

Journal of Selected Topics in Signal Processing 16, 1505–1518. doi: 10.1109/JSTSP.2022.3188113

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020). A Simple Framework for Contrastive Learning of Visual Representations., in *Proceedings of the 37th International Conference on Machine Learning*, (PMLR), 1597–1607. Available at: https://proceedings.mlr.press/v119/chen20j.html

Chen, W., Chang, X., Peng, Y., Ni, Z., Maiti, S., and Watanabe, S. (2023). Reducing Barriers to Self-Supervised Learning: HuBERT Pre-training with Academic Compute., in *Proc. INTERSPEECH 2023*, 4404–4408. doi: 10.21437/Interspeech.2023-1176

Chung, J. S., Huh, J., Mun, S., Lee, M., Heo, H.-S., Choe, S., et al. (2020). In Defence of Metric Learning for Speaker Recognition., in *Interspeech 2020, 21st Annual Conference of the International Speech Communication Association, Virtual Event, Shanghai, China, 25-29 October 2020*, eds.H. Meng, B. Xu, and T. F. Zheng (ISCA), 2977–2981. doi: 10.21437/Interspeech.2020-1064

Chung, J. S., Nagrani, A., and Zisserman, A. (2018). VoxCeleb2: Deep Speaker Recognition., in *Proc. Interspeech 2018*, 1086–1090. doi: 10.21437/Interspeech.2018-1929

Conneau, A., Baevski, A., Collobert, R., Mohamed, A., and Auli, M. (2021). Unsupervised Cross-Lingual Representation Learning for Speech Recognition., in *Interspeech 2021*, (ISCA), 2426–2430. doi: 10.21437/Interspeech.2021-329

Das, R. K., Abhiram, S., Prasanna, S. R. M., and Ramakrishnan, A. G. (2014). Combining source and system information for limited data speaker verification., in *Proc. Interspeech 2014*, 1836–1840. doi: 10.21437/Interspeech.2014-417

Defferrard, M., Benzi, K., Vandergheynst, P., and Bresson, X. (2017). FMA: A Dataset For Music Analysis., in *ISMIR 2017*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database., in *2009 IEEE conference on computer vision and pattern recognition*, 248–255.

Deng, J., Guo, J., Xue, N., and Zafeiriou, S. (2019). ArcFace: Additive Angular Margin Loss for Deep Face Recognition., in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Desplanques, B., Thienpondt, J., and Demuynck, K. (2020). ECAPA-TDNN: Emphasized Channel Attention, Propagation and Aggregation in TDNN Based Speaker Verification., in *Proc. Interspeech 2020*, 3830–3834. doi: 10.21437/Interspeech.2020-2650

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding., in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and*

Short Papers), (Association for Computational Linguistics), 4171–4186. Available at: https://aclanthology.org/N19-1423

Evain, S., Nguyen, H., Le, H., Boito, M. Z., Mdhaffar, S., Alisamir, S., et al. (2021). LeBenchmark: A Reproducible Framework for Assessing Self-Supervised Representation Learning from Speech., in *Proc. Interspeech 2021*, 1439–1443. doi: 10.21437/Interspeech.2021-556

Fan, A., Grave, E., and Joulin, A. (2020). Reducing Transformer Depth on Demand with Structured Dropout., in *International Conference on Learning Representations*. Available at: https://openreview.net/forum?id=SylO2yStDr

Fan, Z., Li, M., Zhou, S., and Xu, B. (2021). Exploring wav2vec 2.0 on Speaker Verification and Language Identification., in *Proc. Interspeech 2021*, 1509–1513. doi: 10.21437/Interspeech.2021-1280

Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., et al. (2016). Domain-adversarial training of neural networks. *The journal of machine learning research* 17, 2096–2030.

Gao, S.-H., Cheng, M.-M., Zhao, K., Zhang, X.-Y., Yang, M.-H., and Torr, P. (2021). Res2Net: A New Multi-Scale Backbone Architecture. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43, 652–662. doi: 10.1109/TPAMI.2019.2938758

Garofolo, J. S. (1993). Timit acoustic phonetic continuous speech corpus. *Linguistic Data Consortium, 1993*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning (chapter 15)*. MIT press. Available at: https://www.deeplearningbook.org/contents/representation.html

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., et al. (2018). Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. Available at: https://arxiv.org/abs/1706.02677

Goyal, P., Mahajan, D., Gupta, A., and Misra, I. (2019). Scaling and benchmarking self-supervised visual representation learning., in *Proceedings of the ieeeCommonVoicef International Conference on computer vision*, 6391–6400. Available at: https://openaccess.thecvf.com/content_ICCV_2019/papers/Goyal_Scaling_and_Benchmarking_Self-Supervised_Visual_Representation_Learning_ICCV_2019_paper.pdf

Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. (2006). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks., in *Proceedings of the 23rd international conference on Machine learning*, 369–376.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition., in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hendrycks, D., and Gimpel, K. (2016). Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*.

Hsu, W.-N., Bolte, B., Tsai, Y.-H. H., Lakhotia, K., Salakhutdinov, R., and Mohamed, A. (2021). HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 29, 3451–3460. doi: 10.1109/TASLP.2021.3122291

Hu, J., Shen, L., and Sun, G. (2018). Squeeze-and-Excitation Networks., in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 7132–7141. doi: 10.1109/CVPR.2018.00745

Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Q. (2016). Deep networks with stochastic depth., in *European conference on computer vision*, 646–661.

Itou, K., Yamamoto, M., Takeda, K., Takezawa, T., Matsuoka, T., Kobayashi, T., et al. (1999). JNAS: Japanese speech corpus for large vocabulary continuous speech recognition research. *Journal of the Acoustical Society of Japan (E)* 20, 199–206.

Izsak, P., Berchansky, M., and Levy, O. (2021). How to Train BERT with an Academic Budget., in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 10644–10652. doi: 10.18653/v1/2021.emnlp-main.831

Jacobs, C., Rakotonirina, N. C., Chimoto, E. A., Bassett, B. A., and Kamper, H. (2023). Towards hate speech detection in low-resource languages: Comparing ASR to acoustic word embeddings on Wolof and Swahili., in *INTERSPEECH 2023*, 436–440. doi: 10.21437/Interspeech.2023-421

Jang, E., Gu, S., and Poole, B. (2017). Categorical Reparameterization with Gumbel-Softmax., in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. Available at: https://openreview.net/forum?id=rkE3y85ee

Jayanna, H., and Mahadeva Prasanna, S. (2009). An experimental comparison of modelling techniques for speaker recognition under limited data condition. *Sadhana* 34, 717–728. doi: https://doi.org/10.1007/s12046-009-0042-9

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., et al. (2021). Highly accurate protein structure prediction with AlphaFold. *nature* 596, 583–589.

Kahn, J., Rivière, M., Zheng, W., Kharitonov, E., Xu, Q., Mazaré, P.-E., et al. (2020). Libri-light: A benchmark for ASR with limited or no supervision., in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 7669–7673. Available at: https://arxiv.org/abs/1912.07875

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., et al. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*. Available at: https://arxiv.org/abs/2001.08361

Kingma, D. P., and Ba, J. (2015). Adam: A Method for Stochastic Optimization., in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Available at: http://arxiv.org/abs/1412.6980

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25.

Lam-Yee-Mui, L.-M., Yang, L. O., and Klejch, O. (2023). Comparing Self-Supervised Pre-Training and Semi-Supervised Training for Speech Recognition in Languages with Weak Language Models., in *INTERSPEECH 2023*, 87–91. doi: 10.21437/Interspeech.2023-1802

Lehečka, J., Psutka, J. V., Smidl, L., Ircing, P., and Psutka, J. (2024). A Comparative Analysis of Bilingual and Trilingual Wav2Vec Models for Automatic Speech Recognition in Multilingual Oral History Archives., in *Interspeech 2024*, 1285–1289. doi: 10.21437/Interspeech.2024-472

Lei Ba, J., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *ArXiv e-prints*, arXiv–1607.

Li, R., Jiang, J.-Y., Li, J. L., Hsieh, C.-C., and Wang, W. (2020). Automatic Speaker Recognition with Limited Data., in *Proceedings of the 13th International Conference on Web Search and Data Mining*, (New York, NY, USA: Association for Computing Machinery), 340–348. Available at: https://doi.org/10.1145/3336191.3371802

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., et al. (2014). Microsoft coco: Common objects in context., in *ECCV 2014*, 740–755.

Lin, W., and Mak, M.-W. (2020). Wav2Spk: A Simple DNN Architecture for Learning Speaker Embeddings from Waveforms., in *Interspeech 2020*, 3211–3215. doi: 10.21437/Interspeech.2020-1287

Ling, S., and Liu, Y. (2020). Decoar 2.0: Deep contextualized acoustic representations with vector quantization. *arXiv preprint arXiv:2012.06659*. Available at: https://arxiv.org/abs/2012.06659

Ling, S., Liu, Y., Salazar, J., and Kirchhoff, K. (2020). Deep contextualized acoustic representations for semi-supervised speech recognition., in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 6429–6433. Available at: https://arxiv.org/abs/1912.01679

Ling, S., Liu, Y., Salazar, J., and Kirchhoff, K. (2020). Deep Contextualized Acoustic Representations for Semi-Supervised Speech Recognition., in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 6429–6433. doi: 10.1109/ICASSP40776.2020.9053176

Liu, A. H., Chang, H.-J., Auli, M., Hsu, W.-N., and Glass, J. R. (2023). DinoSR: Self-Distillation and Online Clustering for Self-supervised Speech Representation

Learning. *arXiv preprint arXiv:2305.10005*. Available at: https://arxiv.org/abs/2305.10005

Liu, A. T., Li, S.-W., and Lee, H.-y. (2021). Tera: Self-supervised learning of transformer encoder representation for speech. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 29, 2351–2366. Available at: https://arxiv.org/abs/2007.06028

Liu, A. T., Yang, S.-w., Chi, P.-H., Hsu, P.-c., and Lee, H.-y. (2020). Mockingjay: Unsupervised speech representation learning with deep bidirectional transformer encoders., in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 6419–6423. Available at: https://arxiv.org/abs/1910.12638

Liu, Y., He, L., and Liu, J. (2019). Large Margin Softmax Loss for Speaker Verification., in *Proc. Interspeech 2019*, 2873–2877. doi: 10.21437/Interspeech.2019-2357

Loshchilov, I., and Hutter, F. (2019). Decoupled Weight Decay Regularization., in *International Conference on Learning Representations*. Available at: https://openreview.net/forum?id=Bkg6RiCqY7

Lugosch, L., Ravanelli, M., Ignoto, P., Tomar, V. S., and Bengio, Y. (2019). Speech Model Pre-Training for End-to-End Spoken Language Understanding., in *Proc. Interspeech 2019*, 814–818. doi: 10.21437/Interspeech.2019-2396

Malladi, S., Lyu, K., Panigrahi, A., and Arora, S. (2022). On the SDEs and Scaling Rules for Adaptive Gradient Algorithms., in *Advances in Neural Information Processing Systems*, eds.S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Curran Associates, Inc.), 7697–7711. Available at: https://proceedings.neurips.cc/paper_files/paper/2022/file/32ac710102f0620d0f28d5d05a44fe08-Paper-Conference.pdf

Martin, A. F., and Greenberg, C. S. (2009). NIST 2008 speaker recognition evaluation: Performance across telephone and room microphone channels., in *Tenth Annual Conference of the International Speech Communication Association*.

Mateju, L., Nouza, J., Červa, P., Zdansky, J., and Kynych, F. (2023). Combining Multilingual Resources and Models to Develop State-of-the-Art E2E ASR for Swedish., in *INTERSPEECH 2023*, 3252–3256. doi: 10.21437/Interspeech.2023-737

McCandlish, S., Kaplan, J., Amodei, D., and OpenAI DotA team (2018). An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*. Available at: https://arxiv.org/abs/1812.06162

Meng, Y., Chou, Y.-H., Liu, A. T., and Lee, H.-y. (2022). Don't Speak Too Fast: The Impact of Data Bias on Self-Supervised Speech Models., in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 3258–3262. doi: 10.1109/ICASSP43922.2022.9747897

Micikevicius, P., Narang, S., Alben, J., Diamos, G. F., Elsen, E., García, D., et al. (2018). Mixed Precision Training., in *6th International Conference on Learning Representations, ICLR 2018*. Available at: https://openreview.net/forum?id=r1gs9JgRZ

Milde, B., and Biemann, C. (2018). Unspeech: Unsupervised Speech Context Embeddings., in *Proc. Interspeech 2018*, 2693–2697. doi: 10.21437/Interspeech.2018-2194

Mitrovic, J., McWilliams, B., and Rey, M. (2020). Less can be more in contrastive learning., in *Proceedings on "I Can't Believe It's Not Better!" at NeurIPS Workshops*, (PMLR), 70–75. Available at: https://proceedings.mlr.press/v137/mitrovic20a.html

Mohamed, A., Lee, H.-y., Borgholt, L., Havtorn, J. D., Edin, J., Igel, C., et al. (2022). Self-supervised speech representation learning: A review. *IEEE Journal of Selected Topics in Signal Processing*. Available at: https://arxiv.org/abs/2205.10643

Nagrani, A., Chung, J. S., and Zisserman, A. (2017). VoxCeleb: A Large-Scale Speaker Identification Dataset., in *Proc. Interspeech 2017*, 2616–2620. doi: 10.21437/Interspeech.2017-950

Nagrani, A., Chung, J. S., Huh, J., Brown, A., Coto, E., Xie, W., et al. (2020). Voxsrc 2020: The second voxceleb speaker recognition challenge. *arXiv preprint arXiv:2012.06867*.

NIST (2016). NIST 2016 Speaker Recognition Evaluation plan. Available at: https://www.nist.gov/itl/iad/mig/speaker-recognition-evaluation-2016

NIST (2018). NIST 2018 Speaker Recognition Evaluation plan. Available at: https://www.nist.gov/itl/iad/mig/nist-2018-speaker-recognition-evaluation

Ott, M., Edunov, S., Baevski, A., Fan, A., Gross, S., Ng, N., et al. (2019). FAIRSEQ: A Fast, Extensible Toolkit for Sequence Modeling., in *Proceedings of NAACL-HLT 2019: Demonstrations*.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., et al. (2022). Training language models to follow instructions with human feedback., in *Advances in Neural Information Processing Systems*, 27730–27744. Available at: https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf

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)*, 5206–5210.

Park, D. S., Chan, W., Zhang, Y., Chiu, C.-C., Zoph, B., Cubuk, E. D., et al. (2019). SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition., in *Proc. Interspeech 2019*, 2613–2617. doi: 10.21437/Interspeech.2019-2680

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library., in *Advances in Neural Information Processing Systems*, . Available at: https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf

Paul, D. B., and Baker, J. (1992). The design for the Wall Street Journal-based CSR corpus., in *Speech and Natural Language: Proceedings of a Workshop Held at Harriman, New York, February 23-26, 1992*.

Peddinti, V., Chen, G., Manohar, V., Ko, T., Povey, D., and Khudanpur, S. (2015). JHU ASpIRE system: Robust LVCSR with TDNNS, iVector adaptation and RNN-LMS., in *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 539–546.

Pepino, L., Riera, P., and Ferrer, L. (2021). Emotion Recognition from Speech Using wav2vec 2.0 Embeddings., in *Proc. Interspeech 2021*, 3400–3404. doi: 10.21437/Interspeech.2021-703

Pironkov, G., Dupont, S., and Dutoit, T. (2016). Speaker-aware long short-term memory multi-task learning for speech recognition., in *2016 24th European Signal Processing Conference (EUSIPCO)*, 1911–1915.

Poddar, A., Sahidullah, M., and Saha, G. (2018). Speaker verification with short utterances: a review of challenges, trends and opportunities. *IET Biometrics* 7, 91–101. doi: https://doi.org/10.1049/iet-bmt.2017.0065

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., et al. (2011). The Kaldi Speech Recognition Toolkit., in *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*, (Hilton Waikoloa Village, Big Island, Hawaii, US: IEEE Signal Processing Society).

Pratap, V., Xu, Q., Sriram, A., Synnaeve, G., and Collobert, R. (2020). MLS: A Large-Scale Multilingual Dataset for Speech Research., in *Interspeech 2020*, 2757–2761. doi: 10.21437/Interspeech.2020-2826

Pu, J., Yang, Y., Li, R., Elibol, O., and Droppo, J. (2021). Scaling Effect of Self-Supervised Speech Models., in *Proc. Interspeech 2021*, 1084–1088. doi: 10.21437/Interspeech.2021-1935

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., et al. (2021). Learning Transferable Visual Models From Natural Language Supervision., in *Proceedings of the 38th International Conference on Machine Learning*, (PMLR), 8748–8763. Available at: https://proceedings.mlr.press/v139/radford21a.html

Radford, A., Kim, J. W., Xu, T., Brockman, G., Mcleavey, C., and Sutskever, I. (2023). Robust Speech Recognition via Large-Scale Weak Supervision., in *Proceedings of the 40th International Conference on Machine Learning*, (PMLR), 28492–28518. Available at: https://proceedings.mlr.press/v202/radford23a.html

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training. *OpenAI blog*.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog* 1, 9.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., et al. (2020). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research* 21, 1–67. Available at: http://jmlr.org/papers/v21/20-074.html

Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. (2020). Zero: Memory optimizations toward training trillion parameter models., in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–16. Available at: https://arxiv.org/abs/1910.02054

Ravanelli, M., Parcollet, T., Plantinga, P., Rouhe, A., Cornell, S., Lugosch, L., et al. (2021). SpeechBrain: A General-Purpose Speech Toolkit.

Rohdin, J., Stafylakis, T., Silnova, A., Zeinali, H., Burget, L., and Plchot, O. (2019). Speaker Verification Using End-to-end Adversarial Language Adaptation., in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 6006–6010. doi: 10.1109/ICASSP.2019.8683616

Rouard, S., Massa, F., and Défossez, A. (2023). Hybrid Transformers for Music Source Separation., in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1–5. doi: 10.1109/ICASSP49357.2023.10096956

The Royal Swedish Academy of Sciences (2024b). The Nobel Prize in Chemistry 2024.

The Royal Swedish Academy of Sciences (2024a). The Nobel Prize in Physics 2024.

Sadhu, S., He, D., Huang, C.-W., Mallidi, S. H., Wu, M., Rastrow, A., et al. (2021). wav2vec-C: A Self-Supervised Model for Speech Representation Learning., in *Proc. Interspeech 2021*, 711–715. doi: 10.21437/Interspeech.2021-717

Salimans, T., and Kingma, D. P. (2016). Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks., in *Advances in Neural Information Processing Systems*. Available at: https://proceedings.neurips.cc/paper_files/paper/2016/file/ed265bc903a5a097f61d3ec064d96d2e-Paper.pdf

Schneider, S., Baevski, A., Collobert, R., and Auli, M. (2019). Wav2vec: Unsupervised Pre-Training for Speech Recognition., in *Proc. Interspeech 2019*, 3465–3469. doi: 10.21437/Interspeech.2019-1873

Shallue, C. J., Lee, J., Antognini, J., Sohl-Dickstein, J., Frostig, R., and Dahl, G. E. (2019). Measuring the Effects of Data Parallelism on Neural Network Training. *Journal of Machine Learning Research* 20, 1–49. Available at: http://jmlr.org/papers/v20/18-789.html

Shorten, C., and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of big data* 6, 1–48.

Simonyan, K., and Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition., in *International Conference on Learning Representations*.

Smith, L. N. (2017). Cyclical learning rates for training neural networks., in *2017 IEEE winter conference on applications of computer vision (WACV)*, 464–472.

Smith, L. N., and Topin, N. (2019). Super-convergence: Very fast training of neural networks using large learning rates., in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, 1100612.

Smith, S. L., Kindermans, P.-J., Ying, C., and Le, Q. V. (2018). Don't Decay the Learning Rate, Increase the Batch Size., in *International Conference on Learning Representations*. Available at: https://openreview.net/forum?id=B1Yy1BxCZ

Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical Networks for Few-shot Learning., in *Advances in Neural Information Processing Systems*, eds.I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, et al. (Curran Associates, Inc.), . Available at: https://proceedings.neurips.cc/paper_files/paper/2017/file/cb8da6767461f2812ae4290eac7cbc42-Paper.pdf

Snyder, D., Chen, G., and Povey, D. (2015). Musan: A music, speech, and noise corpus. *arXiv preprint arXiv:1510.08484*.

Snyder, D., Garcia-Romero, D., Sell, G., Povey, D., and Khudanpur, S. (2018). X-vectors: Robust DNN embeddings for speaker recognition., in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 5329–5333.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* 15, 1929–1958.

Tang, Z., Li, L., and Wang, D. (2016). Multi-task recurrent model for speech and speaker recognition., in *2016 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, 1–4.

Thienpondt, J., Desplanques, B., and Demuynck, K. (2020). The idlab voxceleb speaker recognition challenge 2020 system description. *arXiv preprint arXiv:2010.12468*.

Tjandra, A., Choudhury, D. G., Zhang, F., Singh, K., Conneau, A., Baevski, A., et al. (2022). Improved Language Identification Through Cross-Lingual Self-Supervised Learning., in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 6877–6881. doi: 10.1109/ICASSP43922.2022.9747667

Vaessen, N. (2020). Training Multi-Task Deep Neural Networks with Disjoint Datasets., in *dissertation for Master of Science at KTH royal institute of technology*.

Vaessen, N., and van Leeuwen, D. A. (2022). Fine-Tuning Wav2Vec2 for Speaker Recognition., in *International Conference on Acoustics, Speech and Signal Processing*, 7967–7971. doi: 10.1109/ICASSP43922.2022.9746952

Vaessen, N., and van Leeuwen, D. A. (2022). Training speaker recognition systems with limited data., in *Interspeech 2022*, 4760–4764. doi: 10.21437/Interspeech.2022-135

Vaessen, N., and van Leeuwen, D. A. (2023). Towards Multi-task Learning of Speech and Speaker Recognition., in *INTERSPEECH 2023*, 4898–4902. doi: 10.21437/Interspeech.2023-353

Vaessen, N., and van Leeuwen, D. A. (2025). Self-supervised learning of speech representations with Dutch archival data., in *INTERSPEECH 2025*.

van den Oord, A., Li, Y., and Vinyals, O. (2018). Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*. Available at: https://arxiv.org/abs/1807.03748

van den Oord, A., Vinyals, O., and others (2017). Neural discrete representation learning. *Advances in neural information processing systems* 30. Available at: https://proceedings.neurips.cc/paper/2017/hash/7a98af17e63a0ac09ce2e96d03992fbc-Abstract.html

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*.

Wang, C., Riviere, M., Lee, A., Wu, A., Talnikar, C., Haziza, D., et al. (2021). VoxPopuli: A Large-Scale Multilingual Speech Corpus for Representation Learning, Semi-Supervised Learning and Interpretation., in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, (Online: Association for Computational Linguistics), 993–1003. doi: 10.18653/v1/2021.acl-long.80

Wang, J., Wang, K.-C., Law, M. T., Rudzicz, F., and Brudno, M. (2019). Centroid-based Deep Metric Learning for Speaker Recognition., in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 3652–3656. doi: 10.1109/ICASSP.2019.8683393

Wang, P., and Van Hamme, H. (2023). Benefits of pre-trained mono-and cross-lingual speech representations for spoken language understanding of Dutch dysarthric speech. *EURASIP Journal on Audio, Speech, and Music Processing* 2023, 15.

Wang, Y., Yao, Q., Kwok, J. T., and Ni, L. M. (2020). Generalizing from a Few Examples: A Survey on Few-Shot Learning. *ACM Comput. Surv.* 53. doi: 10.1145/3386252

Wolf, T., Lysandre Debut, Sanh, V., Chaumond, J., Delangue, C., Moi, A., et al. (2020). Transformers: State-of-the-Art Natural Language Processing., in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, (Online: Association for Computational Linguistics), 38–45. Available at: https://www.aclweb.org/anthology/2020.emnlp-demos.6

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Wu, Y., and He, K. (2018). Group normalization., in *Proceedings of the European conference on computer vision (ECCV)*, 3–19.

Yang, S.-w., Chi, P.-H., Chuang, Y.-S., Lai, C.-I. J., Lakhotia, K., Lin, Y. Y., et al. (2021). SUPERB: Speech Processing Universal PERformance Benchmark., in *Proc. Interspeech 2021*, 1194–1198. doi: 10.21437/Interspeech.2021-1775

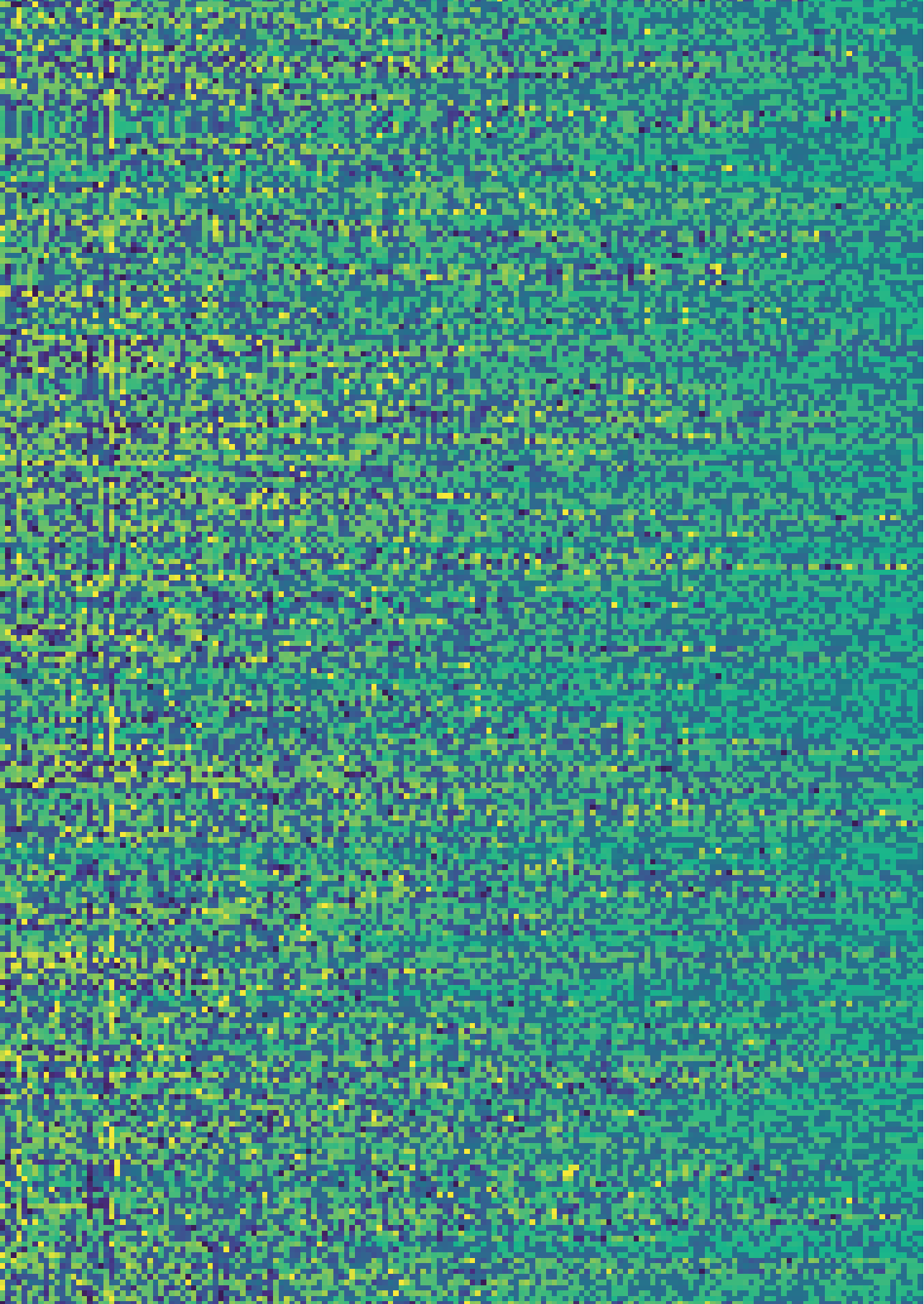
Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. (2019). XLNet: Generalized Autoregressive Pretraining for Language Understanding., in *Advances in Neural Information Processing Systems*, (Curran Associates, Inc.), .

Yuan, J., Cai, X., Zheng, R., Huang, L., and Church, K. (2021a). The Role of Phonetic Units in Speech Emotion Recognition. *arXiv preprint arXiv:2108.01132*.

Yuan, L., Chen, D., Chen, Y.-L., Codella, N., Dai, X., Gao, J., et al. (2021b). Florence: A new foundation model for computer vision. *arXiv preprint arXiv:2111.11432*. Available at: https://arxiv.org/abs/2111.11432

Zhang, C., and Koishida, K. (2017). End-to-End Text-Independent Speaker Verification with Triplet Loss on Short Utterances., in *Proc. Interspeech 2017*, 1487–1491. doi: 10.21437/Interspeech.2017-1608

Zhang, Y., Han, W., Qin, J., Wang, Y., Bapna, A., Chen, Z., et al. (2023). Google USM: Scaling Automatic Speech Recognition Beyond 100 Languages. doi: 10.48550/arXiv.2303.01037

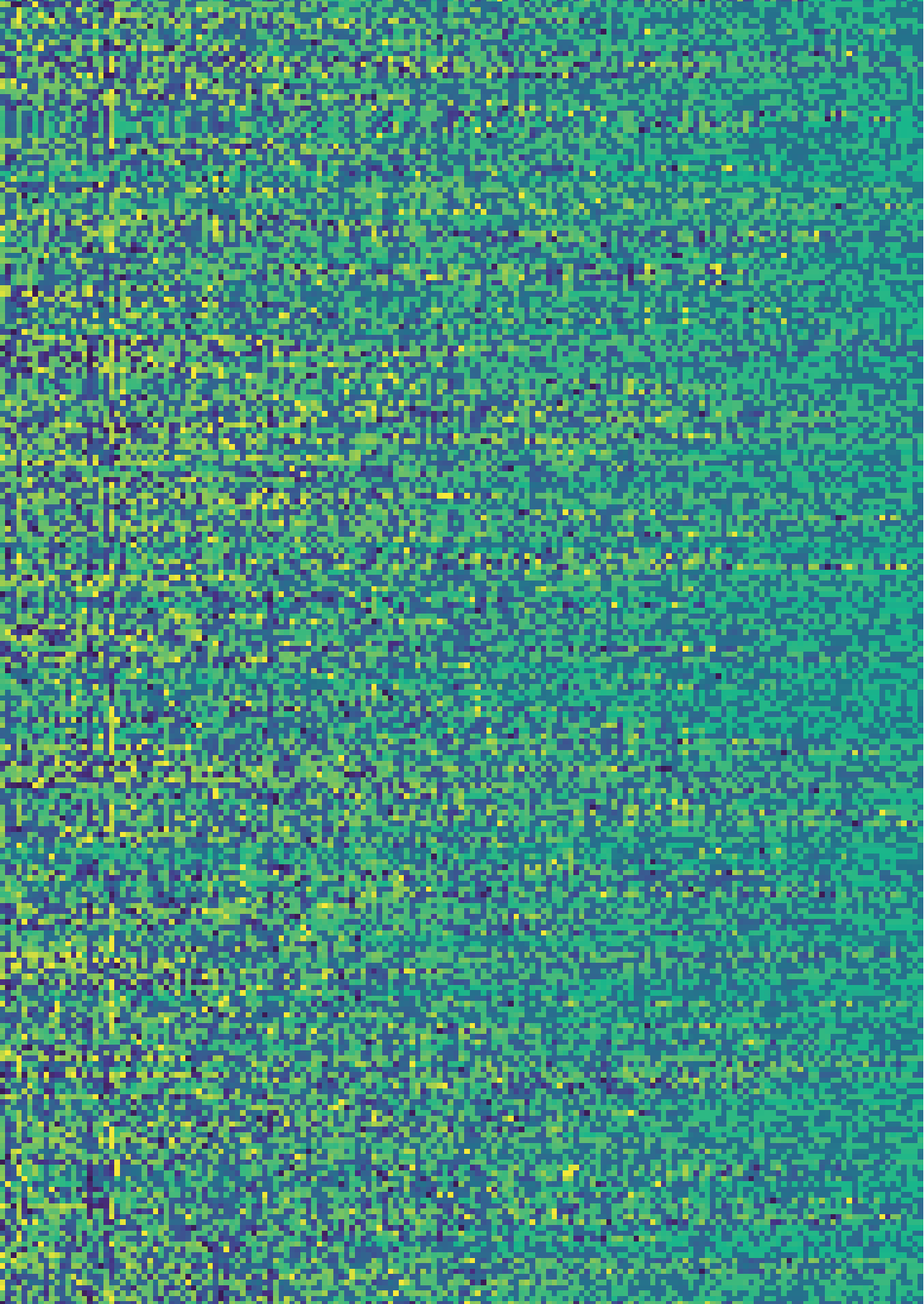


Research data management

This thesis research has been carried out under the research data management policy of the Institute for Computing and Information Sciences at Radboud University, The Netherlands.[27] For each chapter the data and source code availability will be listed:

- **Chapter 2**
  - source code: https://github.com/nikvaessen/w2v2-speaker
  - data:
    - VoxCeleb1: https://doi.org/10.1016/j.csl.2019.101027
    - VoxCeleb2: https://doi.org/10.21437/Interspeech.2018-1929
- **Chapter 3**
  - source code: https://github.com/nikvaessen/w2v2-speaker-few-samples
  - data
    - VoxCeleb1 and VoxCeleb 2
- **Chapter 4**:
  - source code: https://github.com/nikvaessen/2022-repo-mt-w2v2
  - data
    - VoxCeleb1 and VoxCeleb2
    - LibriSpeech: https://doi.org/10.1109/ICASSP.2015.7178964
- **Chapter 5**:
  - source code: https://github.com/nikvaessen/w2v2-batch-size
  - data:
    - VoxCeleb1, VoxCeleb2, Librispeech
    - SUPERB benchmark: https://doi.org/10.1109/TASLP.2024.3389631
- **Chapter 6**:
  - source code:
    - https://github.com/nikvaessen/wav2sr for training
    - https://github.com/nikvaessen/adf for data processing
  - data:
    - Multi-Lingual Librispeech: https://doi.org/10.21437/Interspeech.2020-2826
    - Common Voice: https://doi.org/10.48550/arXiv.1912.06670
    - MUSAN: https://doi.org/10.48550/arXiv.1510.08484
    - Free Music Archive: https://doi.org/10.48550/arXiv.1612.01840
    - Dutch archive data: https://doi.org/10.5281/zenodo.14883498

[27] Available at https://www.ru.nl/en/institute-for-computing-and-information-sciences/research Last accessed on July 15, 2025

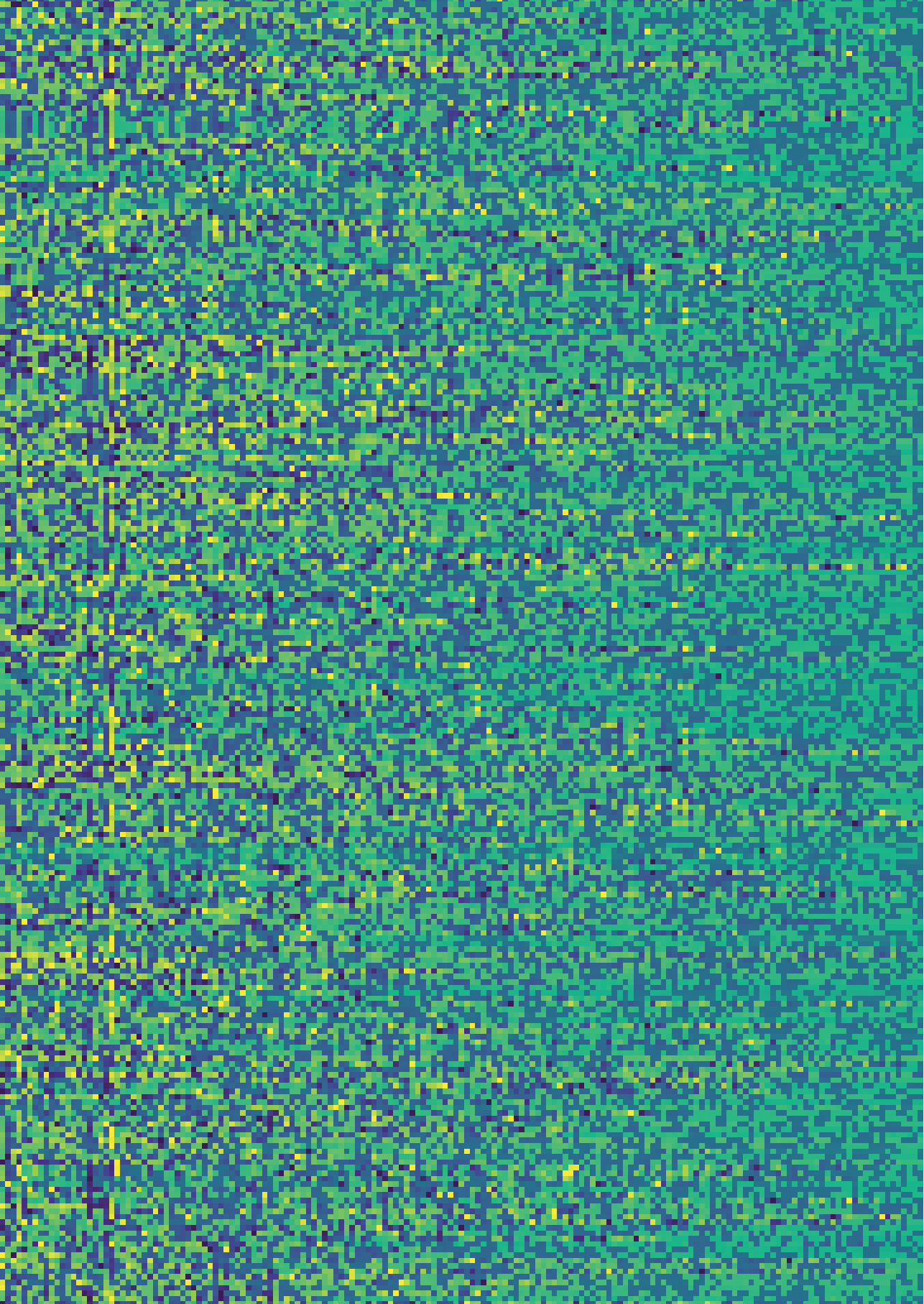


Curriculum vitae

Nik Vaessen began his academic career at Maastricht University, earning a bachelor's degree in Data Science and Knowledge Engineering. During this time, contributions to Jitsi Meet through Google Summer of Code and an internship at Atlassian were made, alongside developing jiwer—a widely-used software package for measuring speech recognition performance—as part of the bachelor thesis.

The academic journey continued with a master's degree in computer science at KTH Royal Institute of Technology in Stockholm, specializing in Deep Learning. A master thesis was completed at Scania's AI team for autonomous driving, complemented by work experience at fintech startup Mysaly.

In 2020, a PhD in multi-task learning for diverse speech technology tasks commenced at Radboud University's Data Science department within the Institute for Computing and Information Sciences. During his PhD, an internship at Amazon AGI was completed, for which research on LLM safety was conducted. He is currently employed at VoxAI, focusing on developing speech recognition technology for drive-thru applications.



Acknowledgements

An adventure cannot be undertaken without mentorship and companionship! I want to thank everyone who took part in this journey, even those who may-not-be-named due to no mistake but my own faulty memory. The PhD experience has its up and downsides, and the people around you make the difference.

First and foremost, I would like to thank David van Leeuwen for his steadfast mentorship and support. Our meetings usually lasted way longer than intended because we simply could not stop talking about the field of speech technology, and you were always ready to answer questions and brainstorm ideas. As a supervisor, you were very hands-off, which I appreciated. You did not mind when I went on a tangent, and you trusted me to deliver eventually. Thank you for guiding me through the PhD process, and I wish you all the best in your own rebellious adventures.

I would also like to give my sincere gratitude to Twan van Laarhoven, Marco Loog, and Gijs van Tulder, who suffered through my occasional, impromptu brainstorm sessions. I would also like to express my gratitude to my manuscript committee in Martha Larson, Hugo Van hamme, and Johan Rohdin for reading and accepting my thesis.

The PhD process is not complete without your fellow colleagues. The data science department had a great atmosphere, with interesting, albeit heavy, conversations during lunch, and of course the nearly-monthly board game sessions cannot be left unmentioned. In alphabetical order, this atmosphere was made possible by Roel Bouman, Gabriel Bucur, Franka Buytenhuijs, David Cicchetti, Marene Dimmendaal, Mohanna Hoveyda, Hideaki Joko, Chris Kamphuis, Alex Kolmus, Zhuoran Liu, Evgenia Martynova, Paulus Meessen, Jelle Piepenbrock, Koert Schreurs, Yuliya Shapovalova, Wieske de Swart, Lin Wouters, and Inge Wortel. I would like to give a special shoutout to Mirthe van Diepen and Emma Gerritse, for which I'm grateful for being able to vent about and discussing personal issues. Finally, I would like to thank my paranymphs Charlotte Cambier van Nooten and Olivier Claessen for humbly accepting this special role in the PhD process, on top of the other things already mentioned!

My stay at Radboud would not have been the same without joining PhD Organisation Nijmegen and the people I met there. Violette Charteau, this manuscript would not have existed without your unrelenting support. A big thanks for providing me with a much-needed social life, in alphabetical order, to Matteo Calzari, Lucía Gómez-Zaragozá, Ayşegül Güneyli, Marc Hermes, Simone Hooijer, Lisa Huis in 't Veld, Jurgen Moonen, Tom van der Most, Daniel Ostkamp, Jessica Ramos-Sanchez, Lucy Spoliar, Etienne Walraven, Kim Wijnant, and Xinyu Zhang.

Before I started this PhD, I was grateful to enjoy the companionship of others outside of Nijmegen. Without their friendship, I would not have been where I am today. Thank you for the role you have played in my life, and for ensuring I ended up where I am today, in alphabetic order, to Sri Datta Budaraju, Maï Cock, Philippe Debie, Esther Kemper, Jan Lucas, Peter Mastnak, Robin Sims, Nina Vogels and Carla Wrede.

I've also had the pleasure to grow professionally due to mentorship of Erik Poromaa at Mysaly and Boris Grozev, Saúl Ibarra Corretgé, and others from the Jitsi Team. Under their supervision I was able to grow my software engineer skills, which provided the backbone of this manuscript. Thank you for contributing to the skills I needed to complete this PhD!

Finally, a heartfelt thank you to my family, who have been there through both the highs and lows of this journey, and especially to my mother, whose absence has been deeply felt throughout the last year and a half. Mam, we hadden allen gewild dat jij dit moment had kunnen meemaken. Je zou ontzettend trots zijn geweest, en met de grootste glimlach hebben genoten van de verdediging. Elke mijlpaal in het leven gaat bitterzoet zijn zonder jouw aanwezigheid. Dit proefschrift is opgedragen in jouw naam. Je zal nooit vergeten worden. Het licht zou *niet* mogen sterven, maar het is helaas toch gebeurd. Geef mijn groeten aan Nika. Rös in vree.

Radboud Universiteit