Self-supervised Keyword Spotting
using Data2Vec Pretraining

SPA10 Master Thesis

Holger Severin Bovbjerg
MSc. in Signal Processing and Acoustics
with Specialisation in Signal Processing and Computing
Aalborg University

02-06-2022

This project report was typeset with the online LaTeX editor Overleaf. The front page brain illustration was found on iStock. Figures have been created using the python package Matplotlib and the online flowchart and diagram maker draw.io.

AALBORG UNIVERSITY
STUDENT REPORT

Department of Electronic Systems
2nd year of study
M.Sc. in Signal Processing and Acoustics
Fredrik Bajers Vej 7B
9220 Aalborg Øst
https://www.es.aau.dk/

Title:

Self-supervised Keyword Spotting using Data2Vec Pretraining

Theme:

SPA10 Master Thesis

Project-period:

February 2022 - June 2022

Project-group:

Group 1071

Participant(s):

Holger Severin Bovbjerg

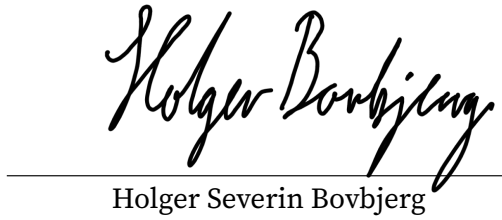
Supervisor(s):

Zheng-Hua Tan

No. of pages: 49
Appendix: 0
Date of completion 02-06-2022

Abstract:

In recent years, the development of accurate deep keyword spotting (KWS) models has resulted in keyword spotting technology being embedded in a number of technologies such as voice assistants. Many of these models rely on large amounts of labelled data to achieve good performance. As a result, most models are restricted to applications for which a large speech dataset can be obtained. Self-supervised learning is a promising area of research which seeks to remove the need for large labelled datasets by leveraging unlabelled data, which is easier to obtain in large amounts. In this thesis, the use of a newly proposed general self-supervised learning framework called Data2Vec is investigated for increasing the performance of KWS models when only a small amount of labelled data is available. A transformer based KWS system is implemented and experiments are carried out on a reduced labelled setup of the Google Speech Commands dataset, to test the benefit of Data2Vec pretraining. It is found that models pretrained using Data2Vec greatly outperform the models without Data2Vec pretraining. The results show that the Data2Vec pretraining can be used to increase the performance of KWS models when only a small amount of labelled data is available for training.



Holger Severin Bovbjerg

Preface

This is a 10th semester master thesis project by Holger Severin Bovbjerg of Signal Processing and Acoustics with Specialization in Signal Processing and Computing at Aalborg University. period from July 2021 to December 2021.

Reading Guide

It is expected that the reader has general knowledge in the area of machine learning, including common neural network model architectures, different layer types, loss functions and optimization methods.

The report follows the rules established by the ISO/IEC 80000 standard. Consequently, SI units are used. In this report, period (.) is used consistently as decimal separator. Citations are made according to the IEEE citation method. All figures without citations are created by the authors for the project. A list of acronyms used in the report and their meanings is found after the table of contents. Figures, tables, equations, and algorithms are inserted and enumerated according to chapter and order of insertion.

The report is structured as follows: First, an introduction to the problem is presented along with a research question. In the next two chapters, the area of deep key word spotting and self-supervised learning a described. Next, an implementation of a keyword spotting system using Data2Vec pretraining is presented. This is followed by a chapter describing experiments and results for Data2Vec pretraining, and finally a conclusion is found.

Acknowledgements

I would like to express my sincere thanks to my supervisor Zheng-Hua Tan, for his always helpful suggestions whenever problems arose, and for many insightful discussions on the topic of self-supervised learning, all of which have been a great help throughout the semester.

Contents

Acronyms

APC Autoregressive Predictive Coding.

ASR Automatic Speech Recognition.

AUC Area Under Curve.

BYOL
Bootstrap Your Own Latent.

CNN
Convolutional Neural Network.

CRNN
Convolutional Recurrent Neural Network.

DCT Discrete Cosine Transform.

DNN
Deep Neural Network.

EMA
Exponetial Moving Average.

FPR False Positive Rate.

GELU
Gaussian Error Linear Unit.

GMM
Gaussian Mixture Model.

GPU Graphics Processing Unit.

GRU
Gated Recurrent Unit.

HMM
Hidden Markov Model.

HPC
High Performance Computing.

KWS
Keyword Spotting.

KWT
Keyword Transformer.

LSTM
Long Short-term Memory.

LVCSR
Large-vocabulary Continous Speech Recognition.

MFCC
Mel-frequency cepstral coefficient.

MLP
Multilayer Perceptron.

MoCo
Momentum Contrast.

MSE
mean squared error.

RAM
Random Access Memory.

ReLU
Rectified Non-linear Unit.

RNN
Recurrent Neural Network.

ROC
Receiver Operating Characteristic.

TCL Time Contrastive Learning.

TPR True Positive Rate.

VICReg
Variation-Invariance-Covariance regularization.

WandB
Weights and Biases.

Chapter

1 Introduction

In the last two decades, deep learning models has proven to be able to solve many advanced tasks such as object recognition in images, reading and generation of text as well as Automatic Speech Recognition (ASR). Historically, deep learning models have mostly been trained to solve specific tasks, benefitting from vast amounts of labelled data used for supervision. As data labelling is a labour intensive process, the need for labelled data has been a constraint for further development in the area of deep learning [1]. Recently, a lot of effort has been put towards the development of more general models which can be reused for multiple tasks, and towards models which can be used when only a small amount of labelled data is available. This movement is partly inspired by the fact that humans do not need to be shown thousands of examples to learn meaningful patterns, and instead learn good representations of the world from observation [2]. Additionally, it has been theorized that humans reuse parts of the brain when learning new tasks [3].

The need for more general models and models which can be trained using small labelled data sets arise in many areas of research. One prevalent example is speech processing, which involves problems such as voice activity detection, speech generation and ASR. In the last couple of years, personal assistants like Google Assistant, Amazon Alexa and Apple's Siri have become a commonplace in day-to-day life. Common for all is that they rely on advanced ASR systems, which are activated by keywords in order to save computational resources when the system is not needed [4]. Keyword activation is done through a so-called Keyword Spotting (KWS) system, which is used to detect keywords in speech. KWS can be seen as a simpler sub-problem of ASR and apart from voice assistants KWS also has a number of other use cases such as speech data mining and phone call routing [5].

While KWS technology has become more and more popular, it is limited by the need for large amounts of labelled training data. Collecting labelled speech data involves human transcription of audio, which is very time-consuming. In order to train models which generalize well, the collected speech data must include many variations of speech, due to the varying nature of speech, such as pitch and dialects [2]. However, obtaining large labelled data sets is not feasible in many cases, e.g. for small languages and local dialects. Therefore, many use cases are restricted to languages where large labelled speech databases are available.

In the search for more generalizable machine learning models that do not rely on large amounts of labelled data, self-supervised learning has shown great promise. Self-supervised learning describes methods in which the data itself is used for supervision, e.g., by removing some part of the data and training the model to fill in the removed part [2]. As a result, self-supervised learning can be used to leverage the vast amounts of available unlabelled data, in contrast to supervised learning which requires labelled data. Self-supervised learning can be utilized to extract good representations of the data, which can then be used for downstream tasks such as classification, by training a smaller model which requires less labelled data to train, using the extracted representations as input [2]. While supervised learning generally achieves the best performance for many tasks, models which are 'pretrained' using a self-supervised learning objective has been shown to achieve similar performance to models which are trained in a purely supervised fashion, sometimes even outperforming them, without needing large labelled data sets [6], [7].

While self-supervised learning is definitely an interesting lead in the search for more general machine learning models, most self-supervised learning methods are modality specific, i.e, the pretext tasks for audio feature encoding are generally not applicable for other modalities such as vision or text [8]. Leading biological theories of human learning suggest that humans use similar learning processes to understand visual inputs as they do for language [9]. Additionally, general model architectures have been shown to perform better than modality-specific architectures [10]. Therefore, learning objectives common to multiple modalities more closely resembles the human learning process, and they might also lead to better performance.

Recently, a framework called Data2Vec was developed with the goal of unifying the self-supervised learning objective for multiple modalities [8]. Data2Vec uses a teacher-student paradigm, with a student model and a teacher model, where the teacher and student networks are identical, and the teacher network weights are an Exponetial Moving Average (EMA) of the student network weights. In the Data2Vec framework, the teacher model first encodes the full input and the student model then encodes a masked input, i.e., a version of input the parts of it have been removed. The learning objective is then for the student model to predict the latent representations of the teacher network from the masked input. In this way, the student model learns to predict the representations of the full input, using only a partial view of the input. Using the Data2Vec framework, the authors were able to outperform previous state-of-the-art self-supervised learning methods in speech, image and text processing.

1.1 Research Question

Data2Vec and many other self-supervised methods have only been investigated for very large models and with very large data sets. Consequently, training these models require numerous high-end Graphics Processing Units (GPUs) as well as many days or even weeks of training. This makes training these models infeasible in many cases, e.g. due to limited time or restricted computing resources. Additionally, for many use cases, such as KWS for voice assistants, it is desired that the models are small [4]. The smallest model used in the Data2Vec study has over 90 million parameters, and is able to produce state-of-the-art results for ASR using a large database for pretraining known as Librispeech [11]. However, many speech processing problems are much simpler than general ASR, and might not require such extensive pretraining nor such a large model, as is the case for KWS. This raises the question of whether the same general learning objective of Data2Vec can successfully be used for simpler tasks like KWS, using significantly smaller models and smaller data sets. Showing that Data2Vec is applicable for small KWS models could potentially help to open the path for self-supervised research using smaller models, making self-supervised learning more accessible for people without access to huge amounts of computing resources. Additionally, it would be a step towards more generally applicable KWS models which can also be used when only sparse amounts of labelled data is available. Based on this, the research question of this master thesis is defined as follows:

> Can Data2Vec pretraining using unlabelled data improve the performance of keyword spotting models, when only a small amount of labelled data is available?

In the following chapters of this master thesis, some background on KWS and self-supervised learning is first presented. This is followed by a description of the implementation of a KWS using Data2Vec pretraining. Next, the experiments for testing performance of the implemented KWS system are described, and their results are presented. Finally, a conclusion of the project is found.

Chapter

2 Keyword Spotting

One of the main features of intelligence is the ability to communicate through speech. With the recent development of machine learning models for speech, speech models are now able to process and understand speech, and can be utilized for a number of speech applications. Voice assistants like Apple's Siri and Google Assistant are examples of such applications which have become a common technology in modern society [4]. Voice assistants incorporate speech models to understand spoken words in order to carry out specific actions, such as turning on the lights or setting up calendar appointments. This is done by running ASR systems which translate audio into sentences, which are then further processed to carry out the desired action. However, such systems are computationally expensive to run, therefore voice assistants use a technology known as KWS to activate the ASR system [4]. KWS can be seen as a sub-problem of ASR, and is the task of identifying specific keywords in audio. KWS systems are significantly less computationally expensive than a full ASR system, and are thus used to limit the use of computing resources for voice assistants, when the full ASR system is not needed. Besides voice assistants, KWS can be used in a number of other applications such as phone call routing and speech data mining [5]. In the following chapter a brief introduction to the state of KWS is given followed by a description of the various elements of deep KWS systems. The purpose of the chapter is to give a general overview of KWS systems.

2.1 Current State of Keyword Spotting systems

Early KWS systems were based on Large-vocabulary Continous Speech Recognition (LVCSR) where a speech signal is decoded into a number of probable phonetic unit sequences and the keyword is then searched in these sequences [4]. A disadvantage of such a system is the need to search a large dictionary, leading to high computational complexity. The keyword/filler Hidden Markov Model (HMM) is an example of a less computationally expensive approach [4]. Here, a keyword HMM is trained to model keywords and a filler HMM is trained to model non-keyword audio segments. Viterbi decoding [12] is then used to find the most probable path in the decoding graph. When the likelihood ratio between the keyword HMM and the filler HMM reaches a predefined threshold, the KWS system is started [13]. The first kind of such models used a Gaussian Mixture Model (GMM) to model acoustic features, whereas modern versions utilize a Deep Neural Network (DNN) [4]. Although keyword/filler HMM approaches are more

lightweight than LVCSR systems, the Viterbi decoding can be computationally expensive.

Recently, KWS systems using DNN outputs directly to determine the existence of keywords in audio have become popular [4]. These systems are known as deep KWS systems and have shown significant improvements in accuracy over keyword/filler HMM approaches. Moreover, they do not need Viterbi decoding and can be scaled to match the available computational resources [4]. A deep KWS system can usually be divided into three parts, namely a feature extractor, a DNN acoustic model and posterior handling as illustrated in fig. 2.1.

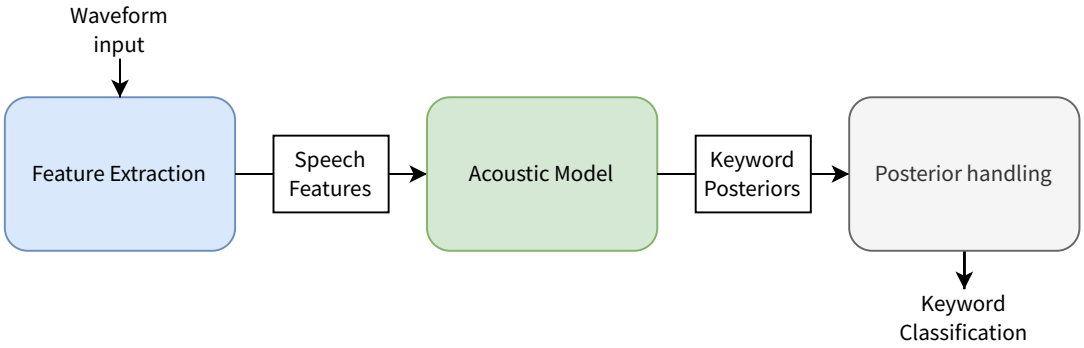


Figure 2.1: Illustration of typical KWS system.

2.2 Speech Feature Extraction

The feature extractor in a KWS system is used to generate a compact representation of the input signal. The purpose of the feature extractor is to reduce the computational complexity of the KWS task, to discriminate the important features of the input, and to make the system more robust to acoustic variations [4]. Typically, a speech sequence $\mathbf{x}$ is translated into a two-dimensional feature representation $\mathbf{X}$ of size of $T \times K$, with T denoting the time dimension and K denoting the feature dimension. Thus, the feature representation $\mathbf{X}$ is composed of T feature embeddings of length K. The speech features $\mathbf{X}$ can be based on a plethora of feature extraction methods such as handcrafted features like spectral, cepstral or time-domain features, trained feature extraction models using neural networks [2], [4] and learned filterbanks [14].

A widely used method for handcrafted speech features is based on log-Mel spectral coefficients and Mel-frequency cepstral coefficients (MFCCs) [4]. These features closely mimic elements of the human auditory system and have shown to be useful for a number of speech related tasks [14]. MFCC features are found by first taking the log-Mel spectrogram of the input sequence $\mathbf{x}$, producing approximately decorrelated features that are suitable for acoustic models [4]. The Discrete Cosine Transform (DCT) transform is then applied to the log-Mel spectrogram to obtain the MFCC features. A general pipeline for extracting Mel-scale features is seen in fig. 2.2.

Neural networks such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have also been used for speech feature extraction by producing feature embedding vectors. As opposed to handcrafted speech features, neural network approaches learn to extract useful features from the raw input audio without the need for human intervention. RNNs like Long Short-term Memory (LSTM) [15] and Gated Recurrent Unit (GRU) [16] networks are known for their ability to model temporal information, and are useful for summarizing variable-length sequences into fixed-length embeddings [4]. Both architectures

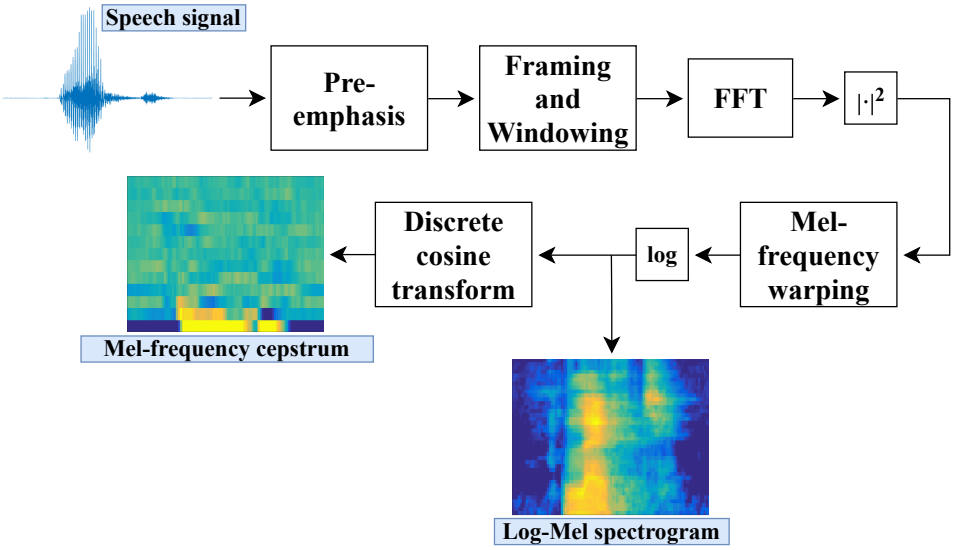


Figure 2.2: General pipeline for extraction of Mel-scale features [4].

have successfully been used for feature extraction in the context of KWS [17], [18]. Approaches using CNNs includes the Wav2Vec model [2] developed by Facebook (now Meta) which makes use of a multilayer CNN model to encode speech features.

Another approach is to use learnable filterbanks to extract useful speech features. The idea behind learnable filterbanks is that the standard Mel-scale filters might not necessarily be optimal for the task at hand. A recent study explored the use of learned filterbanks to extract speech features for KWS and compared them to handcrafted Mel-scale features [14]. The study investigated two learned filterbank approaches. The first approach used a similar method as Mel-scale features, with the Mel-scale filters substituted with a learnable filterbank and the second approach used a gammachirp filterbank with learnable parameters. In both approaches, the filterbanks were trained jointly with a KWS DNN. The general result of the study was that the learned filterbank approaches showed no statistically significant differences compared to using standard Mel-scale features [14].

The overall conclusion is that while neural network based methods have proven to be able to extract useful speech features directly from the raw audio input, handcrafted Mel-scale based features such as MFCC are still widely used and have proven to be competitive with learned feature embeddings [4].

2.3 The Acoustic Model

The following section describes various ways of implementing the acoustic model in a KWS system. An emphasis is put on transformers and the Keyword Transformer (KWT) model, as the KWT model is used in the implementation of chapter 4.

The core element of a KWS system is the acoustic model, which is responsible for modelling keywords from the extracted speech features, and producing posterior probabilities of the keyword being present. The goal of the acoustic model is to provide high accuracy and while being computationally efficient [4]. The first acoustic models for deep KWS relied on fully-

connected neural networks and showed significant improvements over the at the time state-of-the art keyword/filler HMM models [4]. Over time, more advanced approaches such as CNNs and RNNs have substituted the fully-connected neural networks, due to the constant search for increasingly accurate models [4].

Using CNNs for acoustic modelling made it possible to outperform fully connected neural network approaches by efficiently modelling time-frequency correlations. Moreover, residual learning techniques made is possible to make deeper CNNs, with the benefit of increasing the accuracy of the acoustic models [4]. The most well-known example is the seminal Resnet architecture originally proposed in 2015 [19], which has since then been adopted for a plethora of applications.

One drawback of CNNs is their difficulties in modelling long time-dependencies. In contrast, RNNs naturally model time dependencies of temporal sequences, therefore acoustic models based on RNNs perform better for tasks involving long time dependencies. Both LSTM and GRU networks have been explored for KWS acoustic models and have also shown significantly improved performance compared to fully-connected networks [4]. In an effort to bring the best of CNN and RNN models together, Convolutional Recurrent Neural Networks (CRNNs) which combine CNNs and RNNs have been proposed, and have been shown to outperform standalone CNNs and RNNs [4].

Attention mechanisms have also been applied to RNNs to improve performance [4]. The idea behind the attention mechanism is to make the model learn what parts of the input it should focus on, similar to how humans attend to various parts of sentences when communicating. Generally, the attention mechanism in RNNs is implemented by applying some attention function to its hidden state [4]. Given a sequence of hidden states $\{\mathbf{h}_0, \ldots, \mathbf{h}_{T-1}\}$ obtained by encoding the input sequence $\mathbf{x} = \{\mathbf{x}_0, \ldots, \mathbf{x}_{T-1}\}$, a context relevant subset of the hidden state sequence is attended to by applying an attention function $\mathrm{Att}(\cdot)$ to the hidden state vectors in order to produce an attended encoding $\mathbf{A}$. This can be defined as in (2.1).

$$\mathbf{A} = \sum_{t=0}^{T-1} \alpha_t \mathbf{h}_t \tag{2.1}$$

where α_t is the attention weight for time step t found as $\mathrm{Att}(\mathbf{h}_t)$, T is the number of time steps, and $\sum_t \alpha_t = 1$.

2.3.1 Transformer models

While the addition of attention mechanisms to RNN models have improved their performance for KWS, compared to models without attention [4], RNNs have to process the input sequentially, which becomes a limiting factor for long sequences [20]. Lately, a new neural network architecture known as 'transformer' networks [20] has gained much popularity in the machine learning community. Transformers are purely attention based models without any convolutions or recurrence [20]. Instead, they make use of a self-attention mechanism, in order to focus on various parts of the input. Contrary to RNNs, transformer models process the full input sequence at once, making them much more efficient to train, as they allow for better parallelization [20]. The transformer model was initially proposed for machine translation of text, however, it has since been adapted for other modalities such as vision [21] and speech [2], [22].

The self-attention mechanism in transformers works by applying a Scaled Dot Product Attention to the input, making it possible for the model to attend to different parts of the input [20]. The Scaled Dot-Product Attention takes a query $\mathbf{q}$ of dimension d_q, a key $\mathbf{k}$ of dimension d_k and a value $\mathbf{v}$ of dimension d_v as input. The query, key and value vectors are obtained by matrix multiplication of the input with learned weight matrices $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v$. The self-attention weights are found by computing the dot product between the query and key vectors, dividing by $\sqrt{d_k}$ and applying a softmax. The attended output is then found by multiplying the attention weights and the value vector $\mathbf{v}$. In practice, the set of queries, keys and values are all packed into matrices $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ yielding (2.2).

$$\text{Att}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\mathsf{T}}{\sqrt{d_k}}\right)\mathbf{V} \tag{2.2}$$

The main building blocks of the transformer model are multi-head attention blocks and Multilayer Perceptron (MLP) blocks. A so-called multi-head attention is obtained stacking multiple self-attention modules. Multi-head attention allows for models to attend to the input in multiple ways, e.g. one attention head might attend to short-time dependencies and another one might attend to long-term dependencies [23]. Given a multi-head attention block with N attention heads, the output from each attention head is typically concatenated and linearly projected to the expected transformer output dimension, as seen in (2.3).

$$\text{MSA} = [\text{SA}_1; \text{SA}_2; \ldots; \text{SA}_N]\mathbf{W}_{\text{proj}} \tag{2.3}$$

where SA_n denotes the output of the nth attention head and $\mathbf{W}_{\text{proj}}$ is a linear output projection matrix.

Combining multi-head attention with an MLP results in a transformer encoder block. Both the multi-head attention and MLP are followed by a layer normalization, and both have a residual connection. Positional information is given to the model by adding a learnable cosine positional embedding to the input. The full transformer encoder is depicted in fig. 2.3. The attention ability of the transformer model can be scaled, by sequentially stacking multiple transformer encoders or by adding more attention heads [23].

Transformer encoders combined with a small MLP prediction head have been investigated for KWS. The KWT model, uses MFCC spectrogram as input, and achieves state-of-the-art accuracy on the Google Speech Commands datasets [22]. Here, the input spectrogram is divided into a number of patches, which are flattened and used as input tokens for a sequential transformer model with 12 transformer encoders. A learnable CLS embedding is concatenated to the input, yielding an output which embeds the whole spectrogram. This CLS embedding is used as an input for an MLP with a single linear layer, which is used to classify keywords. The KWT uses a Cross-Entropy loss, which is a common choice of loss function for KWS models [4], and the weights are optimized using the Adam optimizer [24]. The model is evaluated on the Google Speech Commands V2 dataset [25] which is a popular benchmark for KWS systems. Using this approach together with some data augmentation, the authors of the KWT paper were able to achieve accuracies up to 98.54 % in 12 keyword setting, and up to 97.74 % in 35 keyword setting.

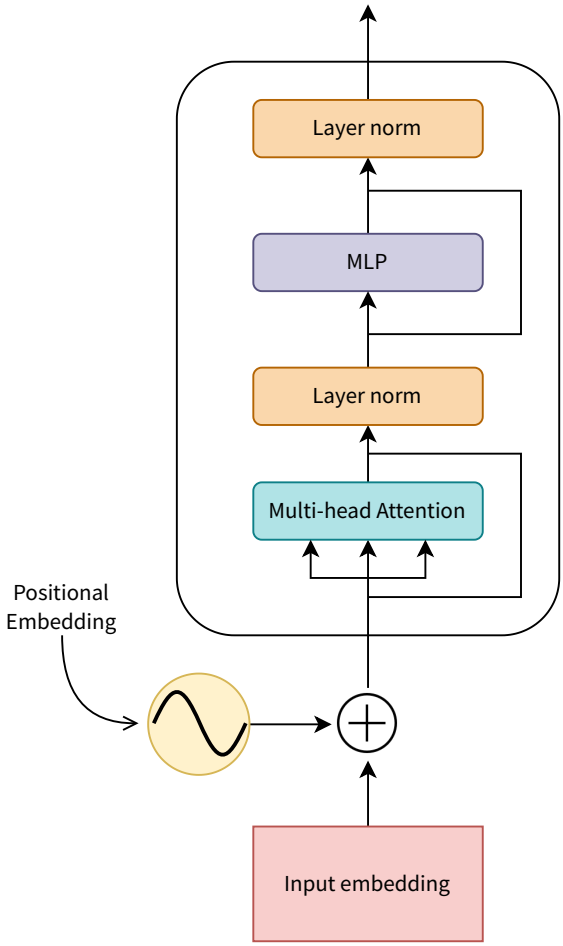


Figure 2.3: Single transformer encoder.

2.4 Posterior Handling

The acoustic model typically outputs a sequence of posteriors $\mathbf{Y} = \left\{\mathbf{y}^{(0)}, \mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(N-1)}\right\}$, with each $\mathbf{y}^{(t)}$ being the posteriors over N_c classes, such that $\mathbf{y}^{(i)} \in \mathbb{R}^{N_\mathrm{c} \times 1}$. The goal of posterior handling is then to choose an appropriate classification from given the sequence of posteriors. This task can be generally carried out in two modes, namely non-streaming and streaming mode.

Posterior handling, in non-streaming mode, describes multi-class classification of input segments, typically with a length of around 1 s, each consisting of a single word [4]. Here, the input segment $\mathbf{X}^{(i)}$ is usually just associated with the highest posterior probability class in the corresponding posterior $\mathbf{y}^{(i)}$. As the input segment $\mathbf{X}^{(i)}$ only contains one word, the non-streaming mode does not have to deal with inter-class segments. This means that the posteriors are usually very peaked for one class, and therefore simply picking the highest probability keyword class generally works well [4].

In a realistic scenario, a KWS system has to dynamically process incoming audio [4]. As a result, non-streaming posterior handling does not suffice for real use cases, as the input segments might not contain the whole keyword. Instead, more advanced methods are used which, given a sequence of posteriors, seek to find the best keyword classification taking all posteriors into account. First, it is common to smoothen the posteriors to generate a new

smoothed posterior $\bar{\mathbf{y}}^{(i)}$, for example by using a moving average [4]. Using the smoothed posterior, keyword classification can be done by using a sensitivity threshold on the posterior probability, or simply by selecting the class with the highest posterior probability. However, in the case that a keyword spans multiple segments, such methods might trigger multiple times for the same keyword. This can be solved by disabling keyword triggering in the KWS system for a short period after a keyword has been spotted [4].

More sophisticated methods for streaming KWS exist for more advanced cases, e.g. in the case of multi-word keywords or when the keyword classification is done using multiple subword units instead of one word. Such methods include lattice search, where the sequence of subword units most similar to the keyword are searched in a lattice, and if the probability of the searched sequence is higher than some threshold, the keyword is classified as present [4].

2.5 Evaluation of Keyword Spotting Systems

As with many other systems, some means of measuring the performance of a KWS system is necessary in order to compare it to other KWS systems. While this is rather obvious, the choice of performance measure is not always obvious. The best performance measure would be a test with relevant end-users, however, as performing such tests is both costly and time-consuming, the use of objective evaluation metrics is most common [4]. Objective evaluation metrics seek to measure the 'goodness' of the system, and should correlate with the subjective end-user evaluation.

One of the simplest yet popular evaluation metrics is the accuracy measure. The accuracy measure is simply the number of correct classifications divided by the total number of classifications. For binary classification (keyword/no keyword) this can be calculated by the sum of true positives and true negatives, divided by the total number of classifications, as seen in (2.4). In the case of multiple keywords, the evaluation metric is commonly computed for each keyword, and the results are averaged [4].

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{2.4}$$

While accuracy is a popular performance metric, one can easily see from (2.4) that many negatives relative to positives, or vice versa, will lead to the performance on one class dominating the accuracy score, which can lead to misleading conclusions. This can become a problem for KWS systems, as the system will most likely hear non-keywords most of the time. E.g., a system which purely outputs 'no keyword' would still get a high accuracy score, without the model being able to spot keywords. However, under the circumstances that the data is balanced between the different classes, accuracy can be a useful performance measure for KWS systems.

Another popular performance measure is the use of Receiver Operating Characteristic (ROC) scores, which is found from the True Positive Rates (TPRs) and False Positive Rates (FPRs) when varying the decision threshold, i.e. the threshold for which a keyword is classified as detected [4]. The TPR is found by dividing the number of false positives with the total number of

positives, as seen in (2.5). Likewise, the FPR is found by dividing the number of false positives by the total number of negatives, as seen in (2.6).

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.5}$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{2.6}$$

By varying the decision threshold for which a keyword is classified as detected, a ROC curve can be made. A number of different ROC curves are depicted in fig. 2.4. Here, a perfect classifier has coordinates $(0, 1)$ and a random classifier will lie on the diagonal line. Taking the Area Under Curve (AUC) of the ROC curve yields a performance metric which equals the probability that a randomly chosen positive example is ranked higher than a randomly chosen negative example [4]. The AUC of the ROC is a value between 0 and 1, where a higher value means a better system. The AUC-ROC score can be used for unbalanced scenarios where the accuracy measure is unsuitable [4].

Two other useful metrics are the precision-recall curve and the F_1-score curve. The precision-recall is build similarly as the ROC curve, by sweeping over a decision threshold replacing TPR and FPR with precision defined by (2.7) and recall being the same as FPR [4].

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.7}$$

The F_1-score is defined from the harmonic mean of the precision and recall, as seen in (2.8). In fig. 2.5 the precision-recall and $F-1$-score curves are illustrated. Like with the ROC curve and AUC of 1 means a perfect classifier.

$$F_1 = \frac{2}{\text{Recall}^{-1} + \text{Precision}^{-1}} \tag{2.8}$$

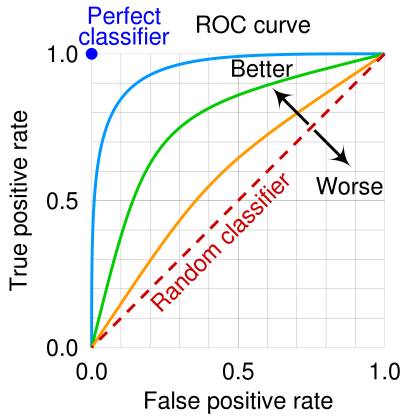


Figure 2.4: Illustration of different roc curves [26].

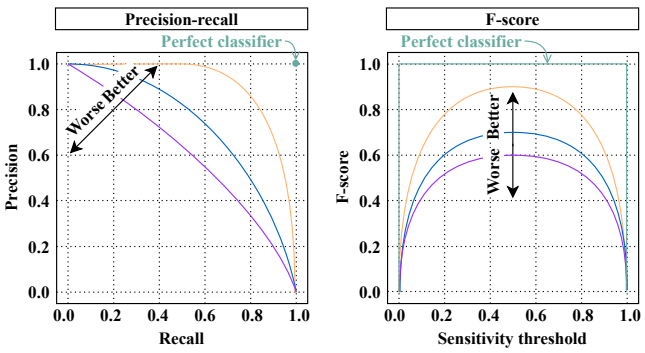


Figure 2.5: Illustration of precision-recall (left) and F_1-score curves [4].

Chapter

3 Self-supervised Learning

In the last couple of years, self-supervised learning has gained much popularity and is seen as a key element in the search for more general AI algorithms. Self-supervised learning describes methods which learn in a supervised fashion by extracting labels from the input. Such learning methods have gained popularity since they do not require data with human annotated labels to train. In a blog post, Yann LeCun, Chief AI Scientist at Meta and one of the leading people in the machine learning field, describes self-supervised learning as being part of 'the dark matter of intelligence', with dark matter being a metaphor for the notion of common sense [1].

In the following chapter, a short introduction to the fundamentals of self-supervised learning is presented, followed by a description of some popular self-supervised learning methods and some problems involved with self-supervised learning. Lastly, a description of the recently proposed general self-supervised framework Data2Vec is found.

3.1 Introduction to Self-supervised Learning

In the early stages of life, babies learn from observing their environment and using these observations to form general models of concepts like objects, gravity and sound [1]. In contrast, many of the most successful machine learning algorithms rely on supervision to learn about data. Supervised learning requires external supervision through data with human annotated labels, which require a substantial amount of time and resources to collect. Self-supervised learning provides a learning mechanism more similar to that of humans, by using the data itself as supervision, thereby learning concepts inherent to the data without needing external supervision [27].

Self-supervised methods are generally used to extract good representations of data, which can then be used to train simple models for different downstream tasks in a supervised fashion without the need for large amounts of labelled data [27]. Therefore, self-supervised methods train models to solve 'pretext' tasks, in which the objective is to make the model learn a good representation of the data, and the actual output of the model during pretraining is of little interest. After pretraining, the model can be 'fine-tuned' to solve downstream tasks where the model is trained to solve a specific task of interest, such as classification problems.

The general idea of self-supervised learning methods is to predict unobserved or hidden parts of the input from an observed or unhidden part of the input [1]. E.g. the input data sample can be augmented in some way, e.g. by adding noise to the data or removing some part of it. The self-supervised task could then be to associate the augmented input with the unaugmented input, or to fill in the removed part [1]. In this way, self-supervised learning methods utilize the data itself to provide supervision and in order for the model to learn a good representation of the data. A number of such methods has been developed, which can generally be put into two categories, namely contrastive and non-contrastive methods.

3.2 Contrastive Learning

The group of contrastive self-supervised methods make use of positive and negative pairs of samples. Positive pairs are data samples which are similar, e.g., two augmented versions of an input sample. Negative pairs are data samples which are not similar, such as two input samples of different categories, e.g. a picture of a cat and a picture of a house. The objective in contrastive learning is to minimize the difference between embeddings of positive pairs and maximize embeddings of negative pairs. Therefore, the model learns to represent positive pairs as similar, while negative pairs are represented dissimilar to each other [28]. A typical contrastive learning pipeline is illustrated in fig. 3.1.

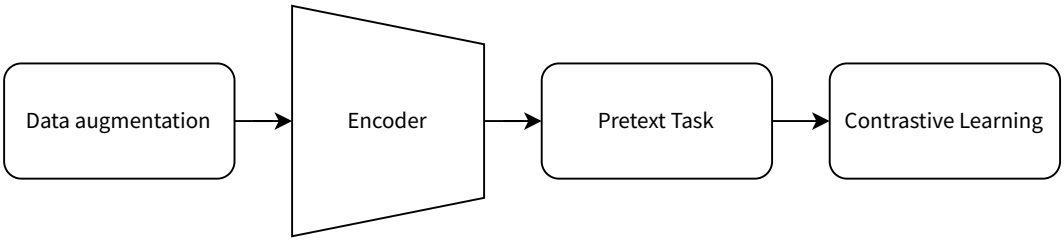


Figure 3.1: Typical contrastive learning pipeline [29].

3.2.1 SimCLR and MoCo

Two common contrastive self-supervised learning methods are SimCLR [30] and Momentum Contrast (MoCo) [28]. Methods like SimCLR and MoCo are so-called *joint embedding architectures*, as they try to produce similar embeddings for different augmentations of the same data [6]. Such architectures are prone to model collapse, i.e. finding a trivial solution such as constant representation, if no mechanism for preventing collapse is used. However, using a contrastive loss can be used to prevent model collapse. Both SimCLR and MoCo use the InfoNCE loss function as the contrastive objective. Here the positive pair is represented as a query and a positive key and a negative pair as a query and a negative key. The InfoNCE loss function supports multiple negative keys and is defined as in (3.1).

$$\mathfrak{L}(q,k^+,k^-) = -\log \frac{\exp(\,(q\cdot k^+)/\tau\,)}{\exp(\,(q\cdot k^+)/\tau\,) + \sum_{k^-\in K}\exp(\,(q\cdot k^-)/\tau\,)} \tag{3.1}$$

where q is query representation, k^+ is a positive sample key representation, k^- is a negative sample key representation, K is the total set of keys, and τ is a temperature parameter which controls the confidence of the model prediction.

The query and keys in (3.1) can be formed in a number of ways. In SimCLR an input sample $\tilde{\mathbf{x}}$ is transformed randomly into two similar samples $\mathbf{x}$ and $\mathbf{x}'$ using data augmentation. Doing this for a batch of input samples yields two matrices $\mathbf{X}$ and $\mathbf{X}'$ as is illustrated in fig. 3.2.

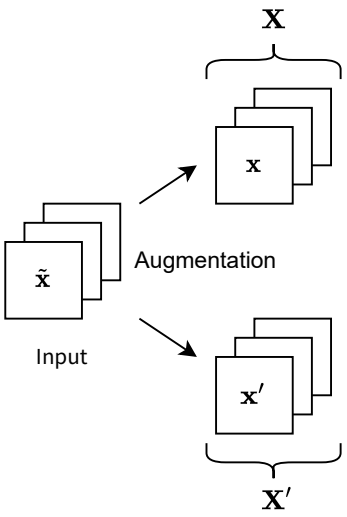


Figure 3.2: Illustration augmentation for generating positive query and key pairs in SimCLR.

After augmentation, two copies of an encoder network f are used to obtain a query representation vector $\mathbf{h}$ and a key representation vector $\mathbf{h}'$ for each sample in $\mathbf{X}$ and $\mathbf{X}'$. These representations are run through a linear projection head h with one hidden layer and a Rectified Non-linear Unit (ReLU) to get representations $\mathbf{Z}$ and $\mathbf{Z}'$ in the space where contrastive loss is applied [30]. The projection is done as it was found to increase model performance compared to using the latent representations directly. In a batch of N samples, a positive query and key pair ($\mathbf{z}$, $\mathbf{z}'$) is generated for each sample. For each positive pair in a batch, the key representations for the other $N-1$ samples in $\mathbf{Z}'$ are used as negative keys in the InfoNCE loss. Using the InfoNCE loss, the weights of f and h are updated through back propagation.

In MoCo the input encodings are generated in the same way as in SimCLR, however the keys are formed using a queuing system [28]. Like SimCLR, MoCo uses two networks of identical architecture, where one used for generating query representations and the other for keys. Initially a batch of N samples, is encoded into queries and keys and the InfoNCE loss is used to update the weights of the query encoder through back-propagation. The keys are then added to a queue of size K. The key encoder weights are an EMA of the query encoder weights in order to achieve consistency in the keys between iterations. When the key encoder queue reaches the maximum number of keys K, the oldest keys are simply popped from the queue. This mechanism makes the available number of negative keys independent of batch size, whereas SimCLR requires a large batch size in order to obtain many negative keys [28]. Figure 3.3 shows a conceptual overview of the SimCLR and the MoCo architecture is seen.

SimCLR and MoCo were originally developed for vision tasks and achieve state-of-the-art performance on a number of computer vision tasks. While both methods were developed for learning of visual representations, some of the same concepts have been adapted for learning speech representations. Jiang *et al.* [31] as well as Al-Tahan and Mohsenzadeh [32] use a similar approach as SimCLR, replacing the image augmentations with time-domain and frequency domain augmentations for audio, in order to learn good speech representation. Both methods

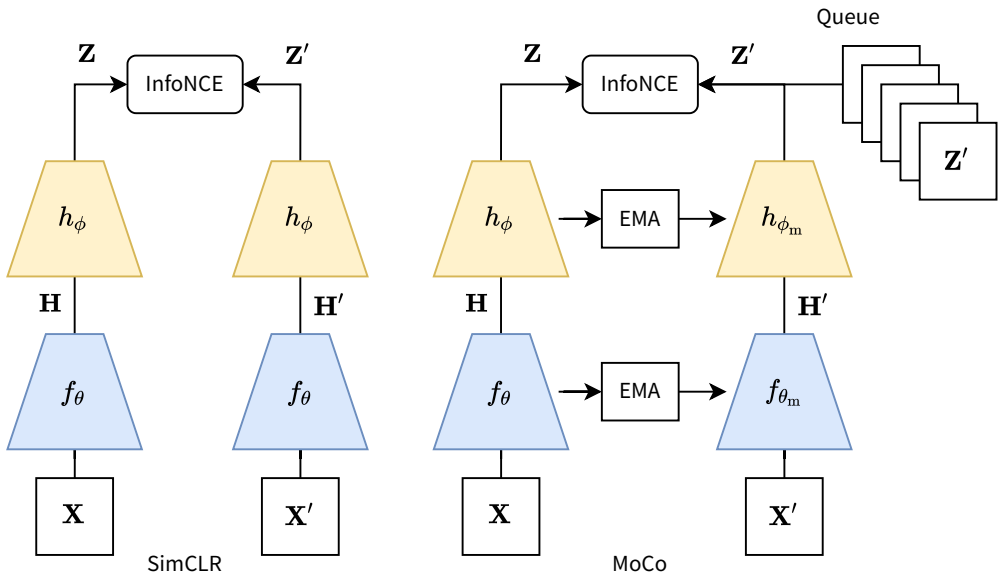


Figure 3.3: Illustration of SimCLR and MoCo architectures. Here θ is the encoder weights, ϕ is the projection weights. θ_m and ϕ_m specifies that the weights are an EMA of θ and ϕ, respectively.

are shown to be able to learn good representations of speech data, which can be used for downstream tasks such as speech recognition.

3.2.2 Wav2Vec

Wav2Vec [2] is another popular contrastive method developed for learning speech representations. The Wav2Vec model competes with supervised models in speech recognition and has shown significantly improved performance when only small amounts of labelled data is available, compared to previous state-of-the-art methods. The Wav2Vec network consists of a feature encoder, a transformer encoder and a quantization module. First, the input audio sequence is put through a CNN feature encoder, which outputs latent speech representations $\mathbf{Z} = \mathbf{z}_1, \ldots, \mathbf{z}_T$ yielding a sequence of T time steps. The number of time steps T is determined by the stride of the CNN feature encoder.

The latent speech representations are used as input to both the quantization module and the transformer encoder. The quantization module maps the feature encoder output $\mathbf{z}$ to a finite set of speech representations, yielding the quantized representation $\mathbf{q}$. The quantization module thus builds speech representations from a fixed number of discrete speech units. Before inputting the latent speech representations to the transformer, some time steps are replaced by a learnable MASK token. This is done by sampling time steps with probability p_{MASK} and masking spans of n future time steps at each sampled time step [2].

The masked latent speech representations are input to a transformer encoder which produces 'contextualized' representations $\mathbf{C}$ of the latent representations, i.e., representation of specific time steps in the context of the others. As the CNN encoders serves as a positional encoding, the addition of a cosine positional encoding is not needed as is in the original transformer [2].

The contrastive task in Wav2Vec is to associate the contextualized representations of a masked time step with the correct quantized representation among a number of candidate representations $\tilde{\mathbf{q}}$ sampled from the other masked time steps. An illustration of the Wav2Vec

architecture is seen in fig. 3.4.

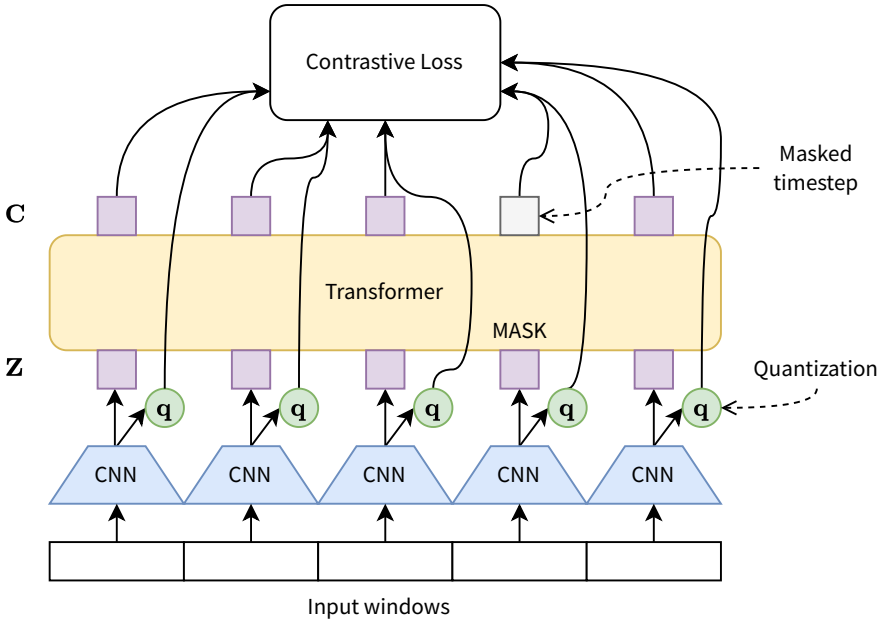


Figure 3.4: Illustration of Wav2Vec architecture.

3.2.3 Other Methods

Other contrastive approaches for speech include the use of a method known as Time Contrastive Learning (TCL) for extraction of deep bottleneck features for speaker verification [33]. The TCL method uniformly divides the audio sequence into N segments and assigns each segment with a class label corresponding to the segment index, i.e., the position of the segment in the audio sequence. A DNN is then used to predict the segment class. In this way, the DNN learns to discriminate time segments of audio. This TCL concept is illustrated for $N = 8$ segments in fig. 3.5. The trained DNN is used to extract bottleneck features for speaker verification, and it has been shown that the resulting bottleneck features perform better than the standard Mel-scale features.

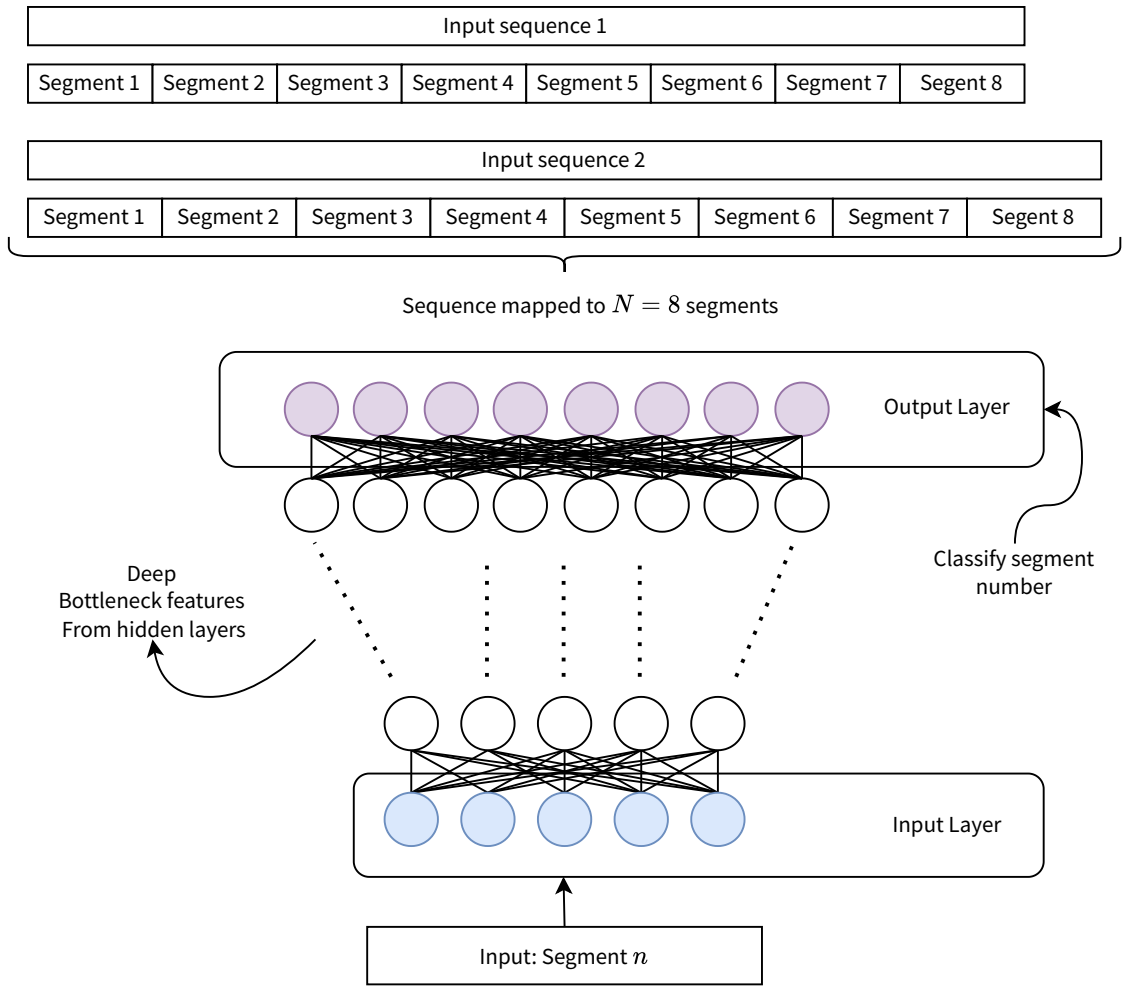


Figure 3.5: Illustration of TCL for extraction of deep bottleneck features.

3.3 Non-contrastive Learning

One problem with contrastive self-supervised learning methods is the need for negative samples. Whereas contrastive learning methods make use of positive and negative examples to learn useful representations of the data, non-contrastive methods do not require contrastive (positive and negative) examples.

3.3.1 Joint embedding methods

Some popular non-contrastive methods include joint embedding architectures like SimSiam [34] and Bootstrap Your Own Latent (BYOL) [7]. Both SimSiam and BYOL where use a Siamese network architecture where two 'branches' with identical networks are used to produce representations from two augmented versions of an input. The objective is then to maximize the similarity between the two representations. As there are no contrastive examples, one might expect the model to simply learn a constant representation, known as model collapse. However, this is can be prevented by the use of some clever tricks.

In the SimSiam model, the two networks have shared weights. Each branch has an encoder and a projection block. On one of the branches, the output representation from the projection network $\mathbf{Z}$ is put through a linear prediction network followed by layer normalization

yielding predicted representations $\mathbf{P}$. On the other branch, $\mathbf{Z}'$ is simply put through a layer normalization and the mean squared error (MSE) loss is computed from $\mathbf{P}$ and $\mathbf{Z}'$. In order to prevent model collapse, SimSiam makes use of a stop-gradient operator on the branch without prediction network, which prevents the gradients from propagating backwards through that branch. An overview of the SimSiam architecture is seen in fig. 3.6.

Instead of sharing weights, BYOL makes use of another trick. Here, one of the encoders are an EMA of the weights from the other branch. The branch with the EMA weights also use a stop-gradient operator as SimSiam. The BYOL is illustrated beside SimSiam in fig. 3.6. The use of an EMA encoder further prevents model collapse. While this has only been shown empirically, it is hypothesized that the mechanism preventing collapse in BYOL, is due to weights of the EMA encoder not being updated in the same direction as the loss gradient [7].

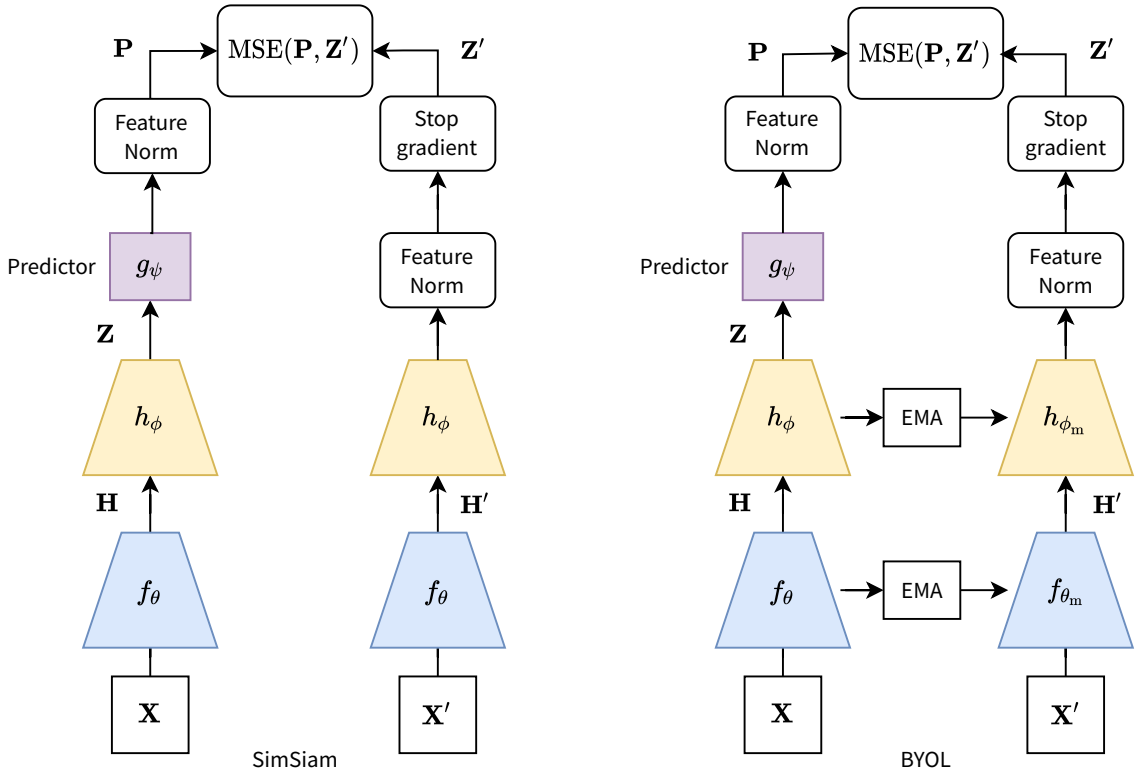


Figure 3.6: Illustration of SimSiam and BYOL architectures.

Both SimSiam and BYOL are able to achieve competitive performance with SimCLR and MoCo, while not needing negative examples. Again, while both methods were originally developed for pretraining computer vision models, similar approaches have been adapted for speech such as BYOL for Audio by Niizumi *et al.* [35]. The primary difference is the augmentation used for generating $\mathbf{X}$ and $\mathbf{X}'$.

3.3.2 Autoregressive Predictive Coding

Autoregressive Predictive Coding (APC) is a non-contrastive method for speech representation learning with unlabelled data, developed by Chung *et al.* [36]. The APC architecture is inspired by language models which predicts the probability of the next token. In APC the self-supervised task is to predict the next n time frames in a spectrogram. More specifically, given time steps $t_1, t_2, \ldots, t_{k-1}$, APC seeks to predict the n future spectrogram frame at time steps

$t_k, \ldots, t_{k+n}$. The objective is then to minimize the L_1-loss between predictions $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_T$ and input $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T$ following (3.2).

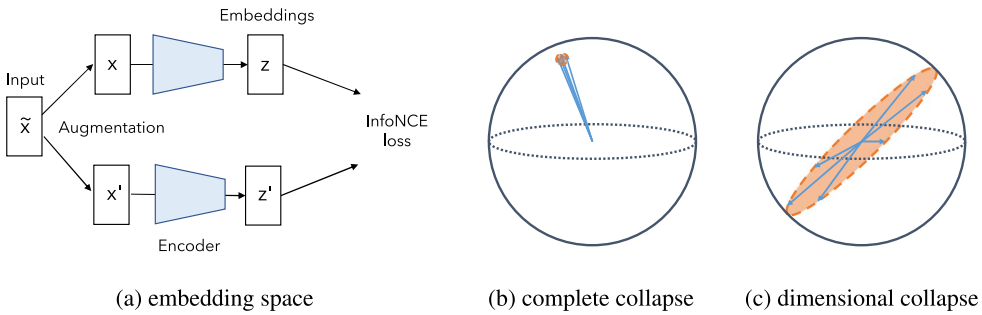
$$\sum_{i=1}^{T-n} |\mathbf{x}_{i+n} - \mathbf{y}_i|_1 \tag{3.2}$$

where T is the number of time steps in the input sequence $\mathbf{x}$.

Although the APC method is rather simple, it has shown good results for a number of downstream tasks, including speaker verification and phoneme classification [36].

3.4 Countering Model Collapse

A prevalent problem in self-supervised learning is when the model finds a trivial and undesired solution to the self-supervised pretext task. One such situation is when the model learns a constant representation, which is known as complete collapse [27]. Most self-supervised methods make use of some regularization to prevent complete collapse, e.g. contrastive methods rely on positive and negative pairs and SimSiam uses a stop-gradient. However, a phenomenon known as dimensional collapse has been observed for both contrastive and non-contrastive self-supervised methods [27]. The driving factors behind dimensional collapse are less obvious and have been studied by Jing *et al.* [27]. In their study, they investigate various causes of dimensional collapse for joint embedding contrastive learning methods, and propose a new method called *DirectCLR* based on their findings. Dimensional collapse can be seen as the model not utilizing the full embedding space to represent the input. An example of complete collapse and dimensional collapse can be seen in fig. 3.7. In (a) a typical joint embedding model using a contrastive InfoNCE loss is seen. The spheres in (b) and (c) represent the embedding space of z in (a) in the case of complete and dimensional collapse, respectively.



(a) embedding space (b) complete collapse (c) dimensional collapse

Figure 3.7: Illustration of: Typical contrastive joint embedding setup (a), complete collapse in embedding space (b), dimensional collapse in embedding space (c) [27].

Jing *et al.* state that there are two main causes of dimensional collapse. One is dimensional collapse by strong augmentation, and another is dimensional collapse by implicit regularization. In the former case, they show that using strong augmentation relative to the capacity of the model leads to dimensional collapse. Using a simple linear model, they show that strong augmentation leads to the embedding vectors falling into a low-dimensional subspace of the full embedding space. Specifically, it is shown that a number of singular values of the embedding covariance matrix fall to zero when increasing the amount of data augmentation [27].

In the case of implicit regularization, Jing *et al.* show that dimensional collapse also happens in the opposite case, i.e., when the model is over-parameterized in comparison to the data augmentation. Here, they empirically show using a two-layer linear model that the weights of the two layers align with each other when only a small amount of data augmentation is used [27].

Based on their findings, Jing *et al.* propose a new method, which applies a contrastive loss directly to the embedding variables. In SimCLR the embedding variables $\mathbf{z}$ and $\mathbf{z}'$ are found by putting the encoder output representations $\mathbf{h}$ and $\mathbf{h}'$ through projection layers as it was found to improve performance. The projection is shown to help prevent dimensional collapse in the SimCLR model [27]. However, Jing *et al.* propose that the weight matrix $\mathbf{W}$ of such a linear projection only has to be diagonal and only has to be low-rank, and they therefore propose the *DirectCLR* method which achieves this, by directly subsampling the embedding vector $\mathbf{h}$. In their *DirectCLR* method, they propose to remove the projection and instead use a normalized subset of the embedding vector $\mathbf{h}$ in the InfoNCE loss such that the input embeddings for computing the loss are found as $\hat{\mathbf{z}} = \mathbf{h}/|\mathbf{h}|$. Using this method, they show better performance than SimCLR on ImageNet [27], without the need for an extra projection layer.

A different study by Bardes *et al.* seek to explicitly combat the model collapse problem by adding regularization terms to the loss function, instead of relying on techniques such as contrastive loss, EMA weight sharing and stop-gradient operators [6]. Their method which they call Variation-Invariance-Covariance regularization (VICReg) uses a loss function of three components, namely an invariance, a variance and a covariance component. VICReg uses a joint embedding architecture, where a batch is augmented to yield $\mathbf{X}$ and $\mathbf{X}'$ which are encoded into representations $\mathbf{H}$ and $\mathbf{H}'$. The encoded representations are then projected into output representations $\mathbf{Z}$ and $\mathbf{Z}'$ as depicted in fig. 3.8.

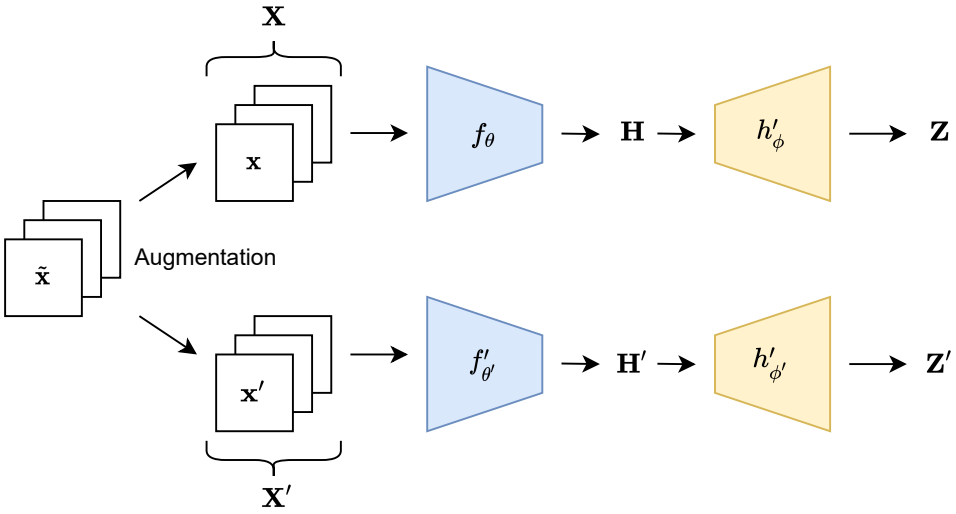


Figure 3.8: Illustration of VICReg joint embedding architecture.

The loss in VICReg are defined as seen in (3.3). As is seen, the loss is made from three components, which promote the following:

1. The invariance term promotes similarity in the output representations $\mathbf{Z}$ and $\mathbf{Z}'$.
2. The variance term promotes variance of the embeddings to be above a certain threshold.

3. The covariance term promotes decorrelation of the embeddings in a batch, preventing informational collapse where the embeddings all vary in the same way.

$$\mathfrak{L}(\mathbf{Z}, \mathbf{Z}') = \lambda s(\mathbf{Z}, \mathbf{Z}') + \mu(v(\mathbf{Z}) + v(\mathbf{Z}')) + \eta(c(\mathbf{Z}) + c(\mathbf{Z}')) \tag{3.3}$$

where s is the invariance loss function, v is the variance loss function, c is the covariance loss function and the scalars λ, μ and η are hyperparameters which scale the importance of each term.

Given two sequences of d-dimensional embeddings $\mathbf{Z} = [\mathbf{z}_1, \ldots, \mathbf{z}_n]$ and $\mathbf{Z}' = [\mathbf{z}'_1, \ldots, \mathbf{z}'_n]$, the invariance loss s is simply defined as the squared euclidean distance between each embedding vector pair, as seen in (3.4).

$$s(\mathbf{Z}, \mathbf{Z}') = \frac{1}{n} \sum_i \left\| \mathbf{z}_i - \mathbf{z}'_i \right\|_2^2 \tag{3.4}$$

The variance loss is found by first finding the regularized standard deviation between each embedding variable following (3.5). A threshold function T defined as is (3.6) is then used, which takes on the value 0 when $S(z^j, \epsilon)$ is above some threshold γ. Here z^j denotes the vector constructed by taking every jth element from all d-dimensional vectors of $\mathbf{Z}$. Lastly, the mean of the threshold function applied to all d dimensions makes up the full variance loss v, as seen in (3.7). This loss essentially encourages the variance of the batch embeddings to be equal to γ.

$$S(\mathbf{x}, \epsilon) = \sqrt{\mathrm{Var}(\mathbf{x}) + \epsilon} \tag{3.5}$$

$$T(\mathbf{z}^j) = \max(0, \gamma - S(\mathbf{z}^j, \epsilon)) \tag{3.6}$$

$$v(\mathbf{Z}) = \frac{1}{d} \sum_{j=1}^{d} T(\mathbf{x}^j) \tag{3.7}$$

Finally, the covariance loss is found from the covariance matrix found as defined by (3.8). The covariance loss is then found as the sum of squared off-diagonal components of the covariance matrix scaled by the number of dimensions d, as seen in (3.9)

$$\mathrm{Cov}(\mathbf{z}) = \frac{1}{n-1} \sum_{i=1}^{n} (\mathbf{z}_i - \bar{\mathbf{z}})(\mathbf{z}_i - \bar{\mathbf{z}})^\mathsf{T}, \quad \text{where} \quad \bar{\mathbf{z}} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{z}_i \tag{3.8}$$

$$c(\mathbf{Z}) = \frac{1}{d} \sum_{i \neq j} \mathrm{Cov}(\mathbf{Z})[i, j]^2 \tag{3.9}$$

Bardes *et al.* test the VICReg model using a ResNet encoder and project the embeddings using three linear layers. VICReg achieves performance comparable to state-of-the-art methods in a number of downstream tasks, relying only on invariance, variance and covariance regularization in the loss function to prevent model collapse. Additionally, the authors show that VICReg regularization can also be incorporated into other methods in order to improve performance.

3.5 The Data2Vec framework

In January 2022 Baevski *et al.* published the Data2Vec framework, which is a non-contrastive method that uses a generalized self-supervised learning task. One of the most promising aspects of the Data2Vec framework is the fact that the self-supervised task can be used for multiple modalities, thereby, potentially opening up the field for more general self-supervised learning. The Data2Vec framework makes use of a student teacher paradigm in which the teacher model receives the full input and the student model receives a masked version of the input, i.e., a version where some input is hidden. The student model then tries to predict the hidden state representation of the teacher model using a linear regression head. This is similar to the learning mechanism in BYOL, however, the targets here are hidden state representations and the self-supervised task is masked prediction similarly to the Wav2Vec model. A general overview of the Data2Vec framework is illustrated in fig. 3.9.

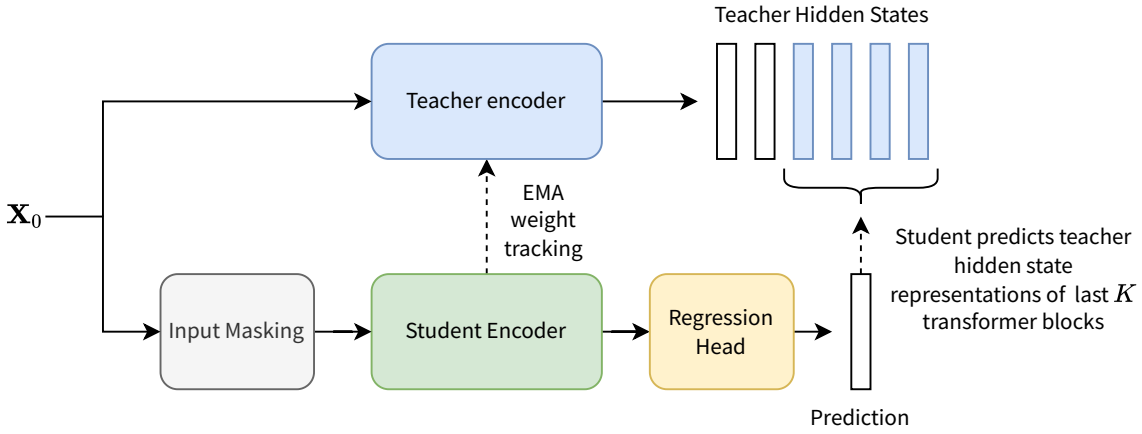


Figure 3.9: Illustration of the Data2Vec framework.

In Data2Vec, the student and teacher networks are identical, with the teacher weights being an EMA of the student weights. Specifically, the teacher weights are updated as seen in (3.10).

$$\Delta = \tau\Delta + (1-\tau)\theta \tag{3.10}$$

where Δ is the teacher weights, τ is an exponential decay constant, and θ is the student weights.

As τ controls how often the teacher model is updated, a linear schedule is used which increases τ from an initial value τ_0 to a target value τ_{end} over n_τ updates. This is done to have the teacher update more frequently in the beginning where the model is random and less frequently when the model has learned a good representation of the data.

In the Data2Vec study [8], standard transformer encoders are used. The goal of the student is then to predict the output representations from the top K transformer blocks of the teacher. It was found that predicting the average of the normalized hidden state representations instead of having individual predictions for each layer performed equally well, while the former is more computationally efficient. As a result, the targets are formed as seen in (3.11).

$$y_t = \frac{1}{K}\sum_{l=L-K+1}^{L}\hat{\mathbf{h}}_t^l \tag{3.11}$$

where y_t is the target at time t, L is the total number of transformer layers and $\hat{\mathbf{h}}_t^l$ is the normalized hidden state representation of the layer l at time t.

Target normalization serves to prevent the model from model collapse, i.e. finding a trivial solution such as a constant representation. While the authors also mention VICReg [6] as a possible regularization measure to prevent model collapse, they found that normalization performed well while not adding additional hyperparameters. In the Data2Vec study, layer normalization is used for vision and text, while the speech model uses instance normalization. The use of instance normalization for speech is motivated by the high correlation between neighbouring representation [8].

The learning objective in the Data2Vec framework is to minimize the difference between the student prediction $f_t(x)$ and the target y_t given by (3.11). They use a smooth L_1 loss as defined in (3.12) for vision and text, and a simple MSE loss for speech. The benefit of using an L_1 loss is that it is less sensitive to outliers [8]. It should be noted that only the masked part is used for computation of the loss.

$$\mathfrak{L}(y_t, f_t(x)) = \begin{cases} \frac{1}{2}\frac{(y_t - f_t(x))^2}{\beta}, & |y_t - f_t(x)| \geq \beta \\ |y_t - f_t(x)|, & \text{otherwise} \end{cases} \tag{3.12}$$

where β is a hyperparameter which controls the transition between a squared loss and L_1 loss.

While the original Data2Vec study does not investigate KWS applications, they do fine-tune the pretrained model for general ASR with low resource setups, i.e. with small amount of labelled data. For all experiments they use two different model sizes, either a Base model with 12 transformer blocks and a hidden dimension of 768, or a Large model with 24 transformer blocks and a hidden dimension of 1024. In their speech experiments, they use a Wav2Vec feature extractor [2], to produce features that are downsampled from 16 kHz to 50 Hz. For pretraining, the input features for the student model are masked in the time domain using the same approach as the Wav2Vec model [2] where fixed length spans of time steps are masked with some probability p_{mask}. The Librispeech speech dataset [11] containing 960 h of speech is used for unlabelled pretraining. Labelled fine-tuning for ASR is done on the Libri-light dataset [37] with different amounts of labelled data varying from 10 min to 960 h. Using Data2Vec pretraining, they achieve an increase in performance for all labelled setups in comparison to two popular speech representation learning models, namely HuBERT [38] and Wav2Vec [2].

Chapter

4 Implementation of Self-supervised Keyword Spotting system

The following chapter describes the design and implementation of a KWS system utilizing the Data2Vec framework for self-supervised pretraining. The KWS system is based on the KWT model by Berg *et al.* [22]. This model is chosen as it fits directly into the Data2Vec framework, as it also uses a transformer encoder. The open-source machine learning framework PyTorch [39] is used for implementing the system in Python. Most of the KWT model implementation is based on a freely available PyTorch implementation [40]. The Data2Vec implementation is based on the code published by the authors in Facebook's Fairseq tool [41], which is a PyTorch based tool for sequence modelling. As the Fairseq tool does not provide as much customization freedom, compared to standard PyTorch, it has been chosen to implement the Data2Vec framework from scratch using the Fairseq implementation as a guideline. The code implementation can be found on `https://github.com/HolgerBovbjerg/data2vec-KWS` and is freely available.

4.1 Features extraction

As mentioned in chapter 2, the first part of a deep KWS system is commonly a feature extraction block. For this part, it is chosen to follow the strategy of the KWT study [22] and use MFCC features as input to the transformer. More specifically, the raw audio is first turned into a Mel-spectrogram using time windows of 30 ms and a stride of 10 ms, after which the DCT transform is applied to obtain the MFCC features. The first 40 MFCC features are used, resulting in an input of $T \times 40$, with T denoting the number of time steps. This feature extraction pipeline is illustrated in fig. 4.1. Implementation of the feature extraction pipeline is done using the open-source Python package *librosa* [42].

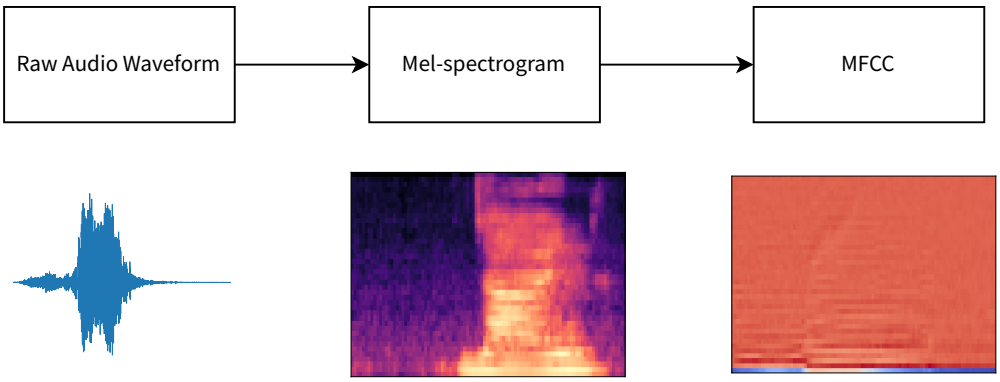


Figure 4.1: Pipeline for turning raw audio into MFCC features

4.2 Acoustic model

The acoustic model architecture is also taken from the KWT model. Here, the input MFCC features $\mathbf{X}$ are first divided into N patches of size $p_t \times p_f$, with p_t being the patch size in the time domain and p_f the patch size in the frequency domain. The patches are flattened to yield vectors of dimension $p_t \cdot p_f$, which are all mapped to an embedding dimension d through a linear projection matrix $\mathbf{W_0} \in \mathbb{R}^{(p_t \cdot p_f) \times d}$ to obtain an input embedding matrix $\mathbf{X}_0$. While the patch size can be chosen to include both time and frequency domain features, the study by Berg *et al.* found that using time-dimension attention, i.e. using patches of size $p_t \times p_f = 1 \times 40$, yielded the best results for KWS [22], thus, the same approach is used here. This effectively means that each patch embedding represents a specific time steps $t_0, t_1, \ldots, t_{T-1}$.

After forming the patch embeddings $\mathbf{X}_0$, a learnable class embedding $\mathbf{X}_{\text{CLS}}$ used to learn global features of the spectrogram is concatenated to $\mathbf{X}_0$ and a learnable positional embedding $\mathbf{X}_{\text{pos}}$ is then added. These steps are summarized in (4.1).

$$\mathbf{X}_0 = [\mathbf{X}_{\text{CLS}}; \mathbf{X}\mathbf{W}_0] + \mathbf{X}_{\text{pos}} \tag{4.1}$$

The feature embeddings $\mathbf{X}_0$ are fed as input to a sequential transformer consisting of 12 transformer blocks with embedding dimension d. Each transformer block is made up by a multi-headed attention block and an MLP block, each followed by layer normalization, as illustrated in fig. 4.2. The MLP blocks consist of a single linear layer followed by a Gaussian Error Linear Unit (GELU) activation, commonly used in transformers [22], [43]. The multi-head attention blocks are simply stacks of self-attention blocks which outputs are concatenated. The concatenated outputs are then linearly projected to the dimension of the transformer block. This is summarized in (4.2).

$$\text{MSA}(\mathbf{X}_l) = [\text{SA}_1(\mathbf{X}_l); \text{SA}_2(\mathbf{X}_l); \ldots; \text{SA}_k(\mathbf{X}_l)]\mathbf{W}_{\text{P}} \tag{4.2}$$

where MSA denotes the multi-headed attention function, $\mathbf{X}_l$ denotes the input of the l'th transformer block, SA is the scaled dot-product attention defined in (2.2) and $\mathbf{W}_{\text{P}}$ is a linear projection matrix, which projects the concatenated self-attentions to the embedding dimension d.

The number of attention heads in the multi-head attention directly relate to the attention capabilities of the model. Thus, scaling the number of attention heads can be seen as a scaling of the model capabilities. In the KWT model the relationship between the number of attentions heads k, and embedding dimension is kept such that $d/k = 64$ following Touvron *et al.* [44].

The output from the transformer blocks consists of $T + 1$ embeddings of the input sequence, with T outputs encoding specific time steps and one CLS embedding encoding the global features of the input. In order to produce keyword predictions, the CLS embedding is fed into an MLP head consisting of a layer normalization and a single linear layer mapping from the embedding dimension to a dimension equal to the number of classes. The full acoustic model is illustrated in fig. 4.2.

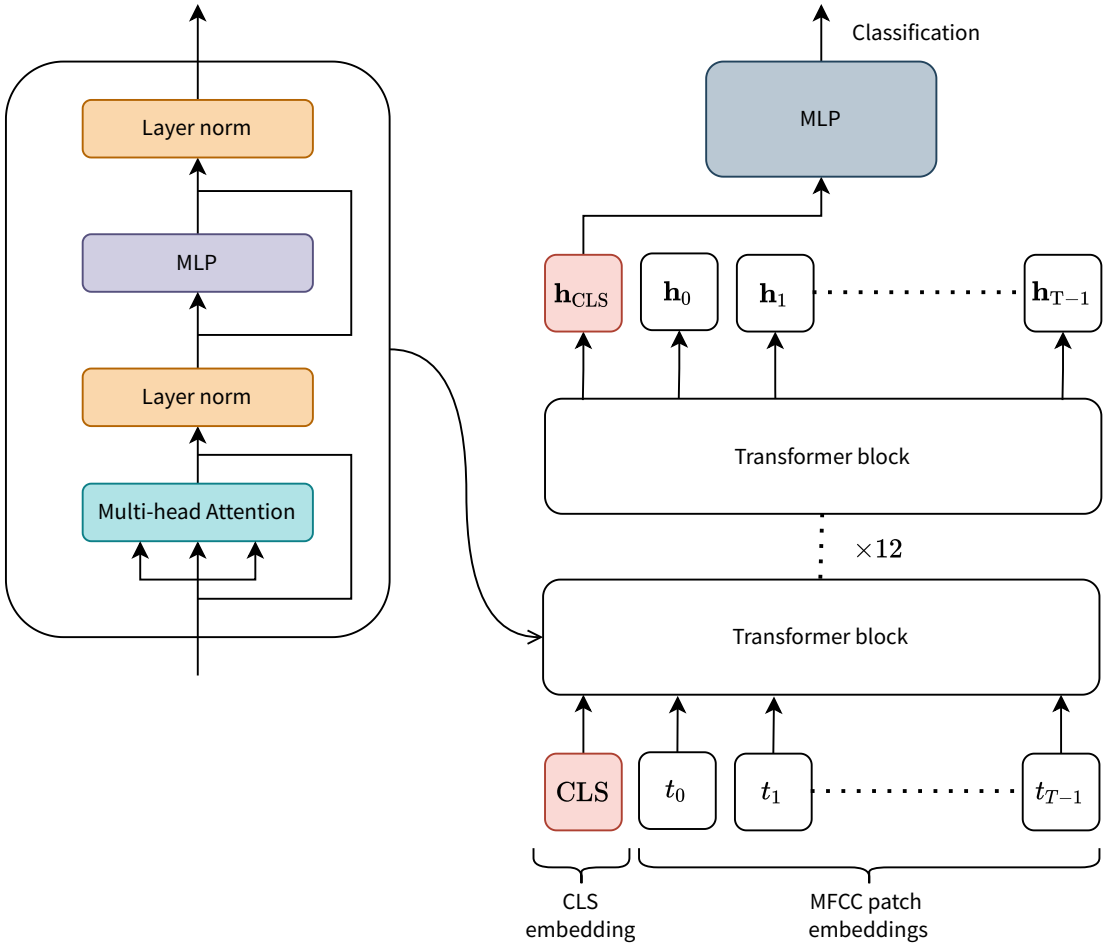


Figure 4.2: Illustration of KWT used for acoustic model.

In the study by Berg *et al.* [22], three different model sizes are created by varying the number of attention heads from 1 to 3, yielding models of varying size as seen in table 4.1. The same approach is adapted here, in order to test how Data2Vec pretraining performs for varying model size.

Table 4.1: KWT model variations.

Model name	Transformer MLP dim.	Encoder dim.	Attn. Heads	Blocks	Parameters
KWT-1	256	64	1	12	607×10^3
KWT-2	512	128	2	12	$2\,394 \times 10^3$
KWT-3	768	192	3	12	$5\,361 \times 10^3$

4.3 Data2Vec Module for Pretraining

This section describes the implementation of a Data2Vec pretraining framework as described in section 3.5 for the KWT model. The implementation of the Data2Vec framework for pretraining of the KWT model can be summarized into the following tasks:

- Provide access to transformer hidden states
- Remove MLP prediction head of KWT model
- Add linear regression layer to last transformer layer output for prediction of teacher hidden states
- Implement a masking strategy
- Implement EMA tracking of student weights for teacher model
- Implement Data2Vec training loop

The above implementation of the above tasks are described in the following section.

The Data2Vec framework requires access to the hidden state representations, and it uses the full transformer output representation instead of the CLS embedding for predictions. Therefore, a slight modification to the KWT model is made for pretraining. First, instead of only outputting the prediction, the hidden state representations are extracted from each layer and added to a list which is given as an output as well. Secondly, the MLP prediction head is removed and replaced by a linear regression head, which takes the full hidden state representation of the last transformer block as an input. This modified model for pretraining is depicted in fig. 4.3.

The masking strategy used in the Data2Vec framework is modality specific. Following the choice for speech masking, a time-domain masking strategy identical to the one used in Wav2Vec [2] is used. Specifically, patches of the input corresponding to time steps are sampled with probability p_{mask}, and the following N_{mask} time steps are replaced by a learnable MASK token embedding. This approach fits well with the chosen patch embedding strategy, as each patch corresponds to one time step. The implementation of the masking algorithm is taken from the Hugging Face *transformer* library's implementation of Wav2Vec [45]. In fig. 4.4, an example of a patch embedding with six spans of 10 masked time steps is seen. Here, the embedding dimension is 64 and the number of time steps is 98. It should be noted that while the CLS input is kept as part of the model, it is never masked, thus, only the patch embeddings are masked.

The student and teacher model in the Data2Vec framework have an identical architecture. While the student network weights are updated using simple backpropagation, the teacher network weights are an EMA of the student network weights. To implement this in PyTorch, an EMA module has been implemented. This module can be used to make a EMA copy of a

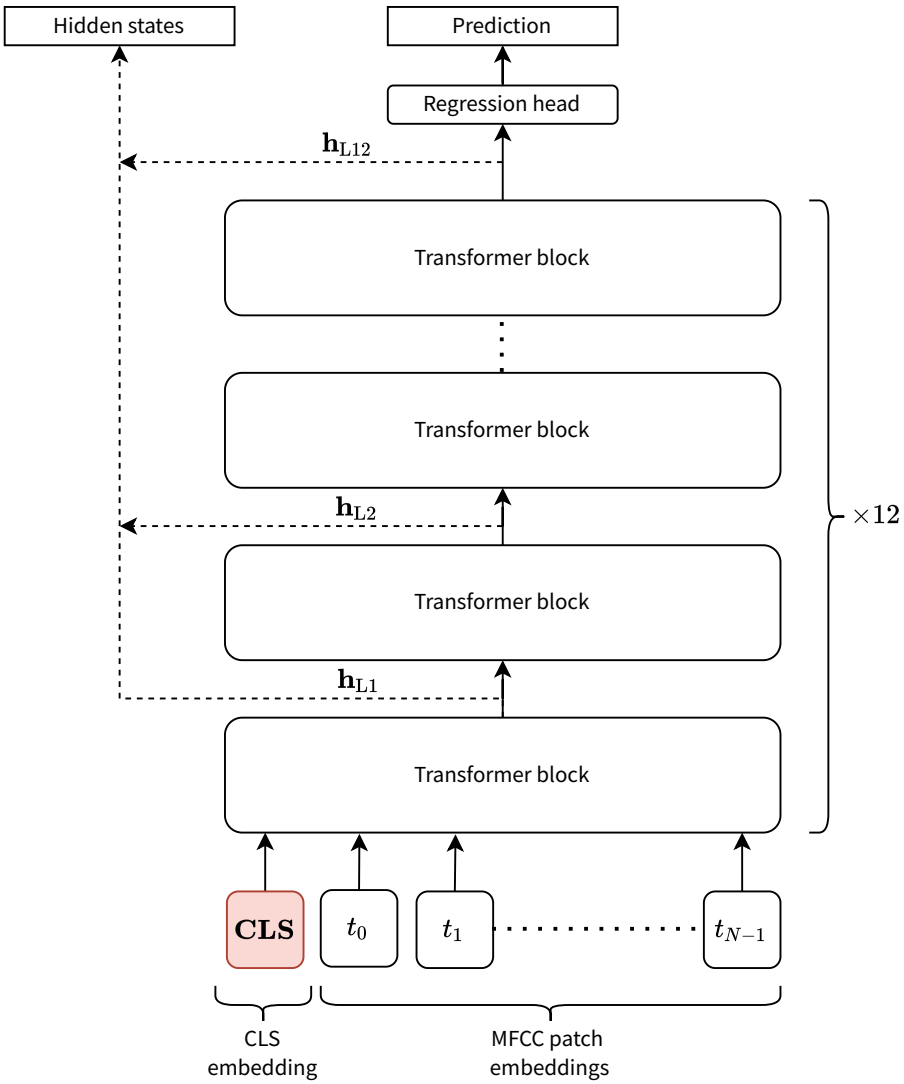


Figure 4.3: Illustration modified KWT encoder for use in Data2Vec framework.

PyTorch model, and keep track of the EMA updates. The teacher model is thus implemented by instantiating an EMA copy of the student model. The teacher model is updated by passing the student model weights to the EMA module each time the student network weights are updated.

Having established a way to extract hidden state representations, masking input and updating the teacher model, the pretraining training loop can be implemented. Before training, a student model equivalent to fig. 4.3 is instantiated. An EMA copy of this model is then created for the teacher model. During training, the inputs are first converted to patch embeddings and a mask is generated for each patch embedding. Masked inputs are then generated by masking the patch embeddings using the generated masks. The masked inputs are then encoded by the student model, and the teacher model is used to encode the unmasked inputs.

After encoding the inputs, the hidden state representations of the transformer layers in the student and teacher model are extracted. The hidden layer representations from the K last transformer layers of the teacher model are normalized and averaged to from the targets. The averaged representations are then normalized again, to yield the final targets. The hidden state representation of the last transformer layer in the student model is put through a linear regression head to generate the target predictions. The loss is then computed from the target

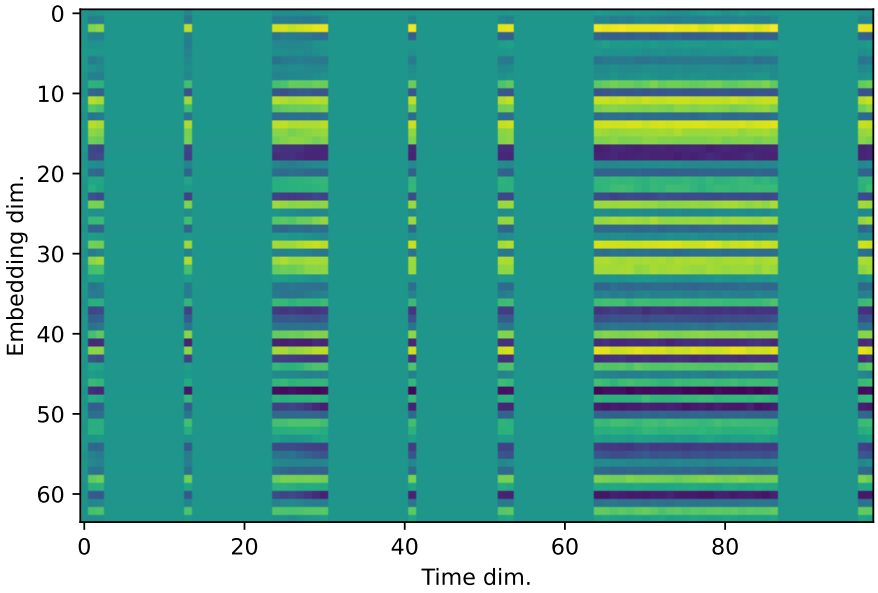


Figure 4.4: Example of masked patch embeddings, with 98 time steps and an embedding dimension of $d = 64$.

and prediction values, and the student model weights are updated through backpropagation. After the student model weights have been updated, they are used to update the EMA teacher model. A PyTorch pseudocode implementation of this training loop is seen in algorithm 1.

After Data2Vec pretraining, the pretrained weights from the modified Data2Vec KWT transformer encoder are loaded into an unmodified KWT model, as seen in fig. 4.2. This model is then trained for keyword spotting using a Cross-Entropy loss as defined in (4.3).

$$\mathfrak{L}_{\text{CE}}(\mathbf{p}, \mathbf{y}) = -\sum_{i=1}^{N} \mathbf{y}_i \log(\mathbf{p}_i) \tag{4.3}$$

where $\mathbf{p}$ is the KWS network classification output, $\mathbf{y}$ is the data labels and N is the number of classes.

Algorithm 1: Pseudocode for PyTorch implementation of Data2Vec pretraining loop for a single epoch.

```
student_model = KWT_encoder()
teacher_model = EMA(student_model)
for batch in dataloader do
    X = to_patch_embedding(batch)
    mask = generate_mask()
    X_masked = X[mask]
    student_hidden_states = student_model(X_masked) # list of student hidden
     states
    teacher_hidden_states = teacher_model(X) # list of teacher hidden states
    target_list = [normalize(h) for h in teacher_hidden_states[-K:]]
    target = normalize(sum(target_list)/len(target_list))
    student_last_encoder_layer_out = student_hidden_states[-1]
    prediction = regression_head(student_last_encoder_layer_out)
    target, prediction = target[mask], prediction[mask] # only masked
     timesteps used for loss
    loss = criterion(prediction, target)
    loss.backward()
    optimizer.step()
    teacher_model.ema_step(student_model)
```

CHAPTER

5 Experiments and Results

In order to evaluate the performance of the implemented self-supervised KWS system, described in chapter 4, a number of experiments have been carried out. In the following chapter, the setup for carrying out experiments is first described, followed by a description of the experiments and the corresponding results. Lastly, an interpretation of the results is given.

5.1 Computing Resources

Training machine learning algorithms require a substantial amount of operations to run. Therefore, one of the main limitations to machine learning is the available computing resources. While it might be feasible to train some smaller models on a modern laptop, most work loads require access to GPUs resources, which can be used to accelerate most machine learning work loads through heavy parallelization. Running PyTorch models on NVIDIA GPUs resources is rather straight forward, as it supports CUDA [39], [46].

Aalborg University has a number of resources available for machine learning work loads. One is the CLAAUDIA's AI Cloud [47] High Performance Computing (HPC) cluster, which contains two DGX-2 servers. These servers contain a total of 32 NVIDIA V100 GPUs with 32 GB Random Access Memory (RAM) each. The NVIDIA V100 GPU are currently some of the most advanced server GPUs available, and makes it possible to train very large work models. In order to run experiments for this project, access has been granted to run work loads on CLAAUDIA. However, as the CLAAUDIA resources are in very high demand, a score-based scheduling system is used to achieve a fair distribution of resources among the users. While this makes it great for running large machine learning pretested works loads with a known runtime, it is not great for development and testing of code, as resources are often times not available. Therefore, other available resources have been utilized, namely CLAAUDIA's Strato Cloud [48]. Here, it is possible to create a virtual machine with some predefined specifications. Recently, it has become possible to allocate GPU resources. The GPU resources available are NVIDIA T40 GPUs which have 16 GB RAM. While the T40 GPUs is smaller and slower than the v100 on CLAAUDIA AI Cloud, it is sufficient for the workloads of this project. Training the largest KWT model on an Nvidia T40 takes approximately 10 h.

5.2 Experiment Tracking

An important element of machine learning research is tracking of performance measures, both in terms of model performance and in terms of resource utilization. Often times, multiple 'runs' of varying settings are carried out, and the metrics are then saved in arrays or lists. In many cases, this is done manually, which can become cumbersome for many runs. Many runs can also take several hours to complete. Additionally, the runs might take unnecessarily long if the computing resources are not fully utilized. Therefore, many hours can be wasted if no monitoring is in place.

To solve this problem, the Weights and Biases (WandB) service has been used. WandB is an online service which provides a python package and a web service that makes it possible to log metrics of runs in real-time. This includes information about model performance, time per iteration, run summaries and utilization of computing resources. The use of WandB has been a great help throughout the development phase in order to monitor potential errors in the model, and to log and analyse experiment results. An overview of the WandB web interface can be seen in fig. 5.1.

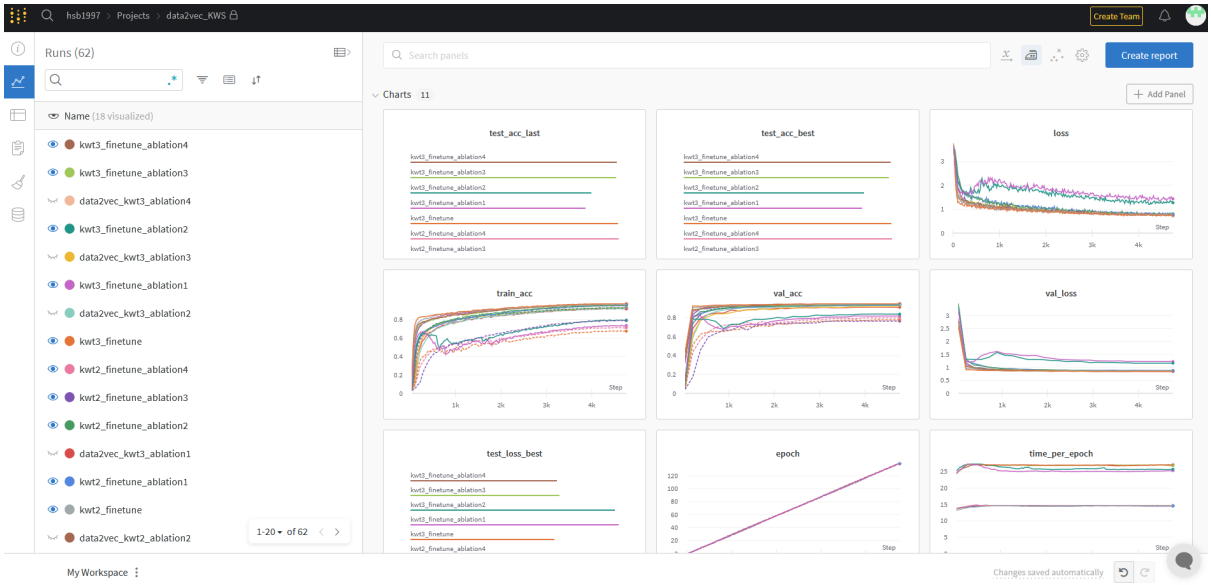


Figure 5.1: Screenshot of Weights and Biases web interface.

5.3 Dataset

The implemented KWS system is evaluated on the open Google Speech Commands V2 dataset [25], which is also used in the original KWT study [22]. This dataset is chosen as it has become a standard for evaluation of KWS systems, and because it has a more moderate size than some alternatives, such as the enormous Librispeech and Libri-light databases used in the Data2Vec study. The Speech Commands V2 dataset consist of 105 829 labelled keyword sequences of clean audio with a duration of 1 s. Each keyword is sampled at 16 kHz and consist of a total of 35 different keywords. The fact that all audio sequences are all approximately 1 s long, also makes data preprocessing simpler, compared to data sets with large variance in the sequence lengths.

The full set of audio sequences in the Speech Commands V2 dataset are originally split into

a training, validation and test set. As the goal is to investigate if Data2Vec pretraining on unlabelled data can improve model performance when only a small amount of labelled data is available, a reduced label version of the Speech Commands V2 dataset is created to simulate a sparse labelled setup. This is done by further splitting the training set 80:20 into a pretraining part used for Data2Vec pretraining set, and a training set used for baseline and fine-tuning. The resulting splits and their number of examples are summarized in table 5.1.

Table 5.1: Reduced label Google Speech Commands V2 splits.

Split	No. keyword examples
Pretrain	67 731
Train	16 932
Validation	10 583
Test	10 583

5.4 Experiments

The experiments are carried out for each of the KWT model variations, which are summarized in table 4.1. This is done to gain insight into the influence of model size on model performance. First, all three KWT variations are trained on the reduced label Speech Command training set to produce baselines. This is done following the same approach as the KWT study [22], with the exception that time-dimension augmentations are removed for computational speedup, as it was found that the computation of the MFCC increased training time significantly. Instead, the MFCC spectrograms are precomputed and loaded into memory and only SpecAugment augmentation [49] is applied during training. As data augmentation can be seen as a way of synthetically adding more data, removing time-domain augmentation also serves to promote data sparsity, potentially making the keyword spotting task more difficult to learn. The model is trained for 140 epochs using a batch size of 512. The weights are optimized using the AdamW optimizer [50] with a two-step learning schedule where the learning rate is initially 'warmed up' by linearly increasing it from $eta_0 = \eta_{\max}/(\text{batch size} \cdot n_{\text{epochs}})$ to $\eta_{\max}$ over the first 10 epochs, after which it follows a cosine annealing schedule [51] for the remaining 130 epochs, as seen in fig. 5.2.

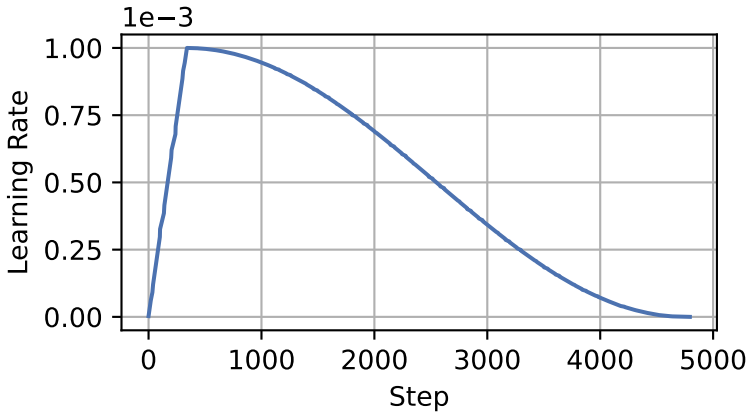


Figure 5.2: Learning rate schedule for baseline.

A weight decay of $\lambda = 0.1$ is also used, which serves to prevent overfitting by penalizing large

weight values [52]. Additionally, a label smoothing of $\epsilon = 0.1$ is applied to the targets. Label smoothing adds noise to the labels to model the uncertainty of the labels [53]. This is done by replacing the 0 and 1 labels with $\epsilon/N - 1$ and $1 - \epsilon$, respectively [53].

A simple classification accuracy metric is used as the KWS performance metric, for evaluation of the KWS system. The accuracy metric has the downside that it can be deceiving for imbalanced data sets, however, as the Speech Commands V2 dataset is rather balanced in terms of different keywords, accuracy is a useful measure of system performance.

In fig. 5.3 the training curves for the three KWT model variations are seen. Here it is seen that all three models achieve a final validation accuracy score just under 80 %. Interestingly, all models perform very similar, although it might be expected that the larger models perform worse for sparse labelled setups as larger transformers often need to be trained on large amounts of data [22].

The baseline results on the test set for all three KWT variations are seen in table 5.2. All three models achieve performance comparable to the training and validation set scores, indicating that the models all generalize well. However, none of the baseline achieve anywhere close to the accuracies achieved on the full training set in the original study, where the models all scored over 96 % accuracy on the test set. Some loss in accuracy can be explained by the fact that time-series augmentation is not used here, however, when training on the full training set, a test set accuracy of 95 % was achieved for the KWT-1 model without using time-domain augmentation. Thus, most of the drop in performance can be attributed to the reduced amount of labelled data available in the reduced training set.

Table 5.2: Summary of baseline results for different KWT variations when trained on the reduced training set of Google Speech Commands.

Model	Test accuracy
KWT1-baseline	0.743
KWT2-baseline	0.7869
KWT3-baseline	0.7578

5.4.1 Data2Vec Pretraining

With the baseline results established, the next step is pretraining using Data2Vec. The goal of Data2Vec pretraining is to learn good representations of the data without use of data labels. The Data2Vec pretraining is done on the set aside pretraining set of the Google Speech Commands V2 dataset, whereafter the pretrained model is trained on the reduced training set used for the baseline results. Usually, pretraining for learning speech representations is done on much larger datasets, with each sample containing full spoken sentences of multiple words, as is the case in the original Data2Vec study [8]. In the Speech Commands dataset, all the samples contain only one keyword and are approximately of the same length. As a result, the learned representations will not be as 'rich' as those learned from pretraining on larger datasets such as Librispeech, which means the learned representations probably won't generalize as well for new data. However, this is deemed not to be a problem, as the main goal is to test whether Data2Vec pretraining can increase the KWS performance when only a small amount of labelled data is available, and due to the fact that fine-tuning is done on similar data from the same dataset.

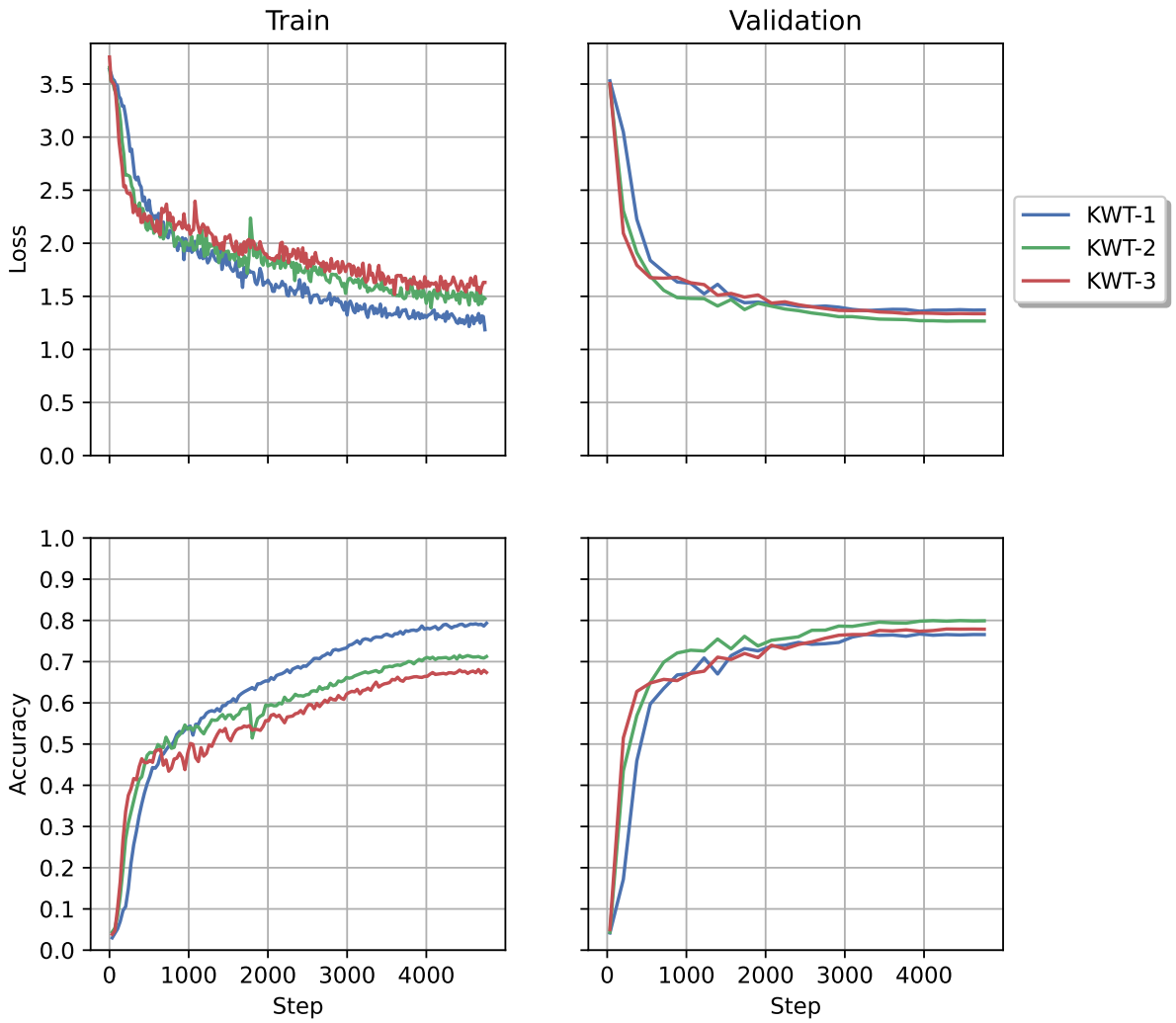


Figure 5.3: Baseline training graphs for all three KWT models.

In the Data2Vec framework, a number of pretraining hyperparameters have to chosen, such as the EMA decay for the teacher model parameters and the masking strategy for masking the student model input. Most of the hyperparameters for pretraining are chosen according to the settings for used speech in the original Data2Vec study [8], with some slight alterations.

The EMA decay is initially set to $\tau_0 = 0.999$ and then linearly annealed to $\tau_e = 0.9999$. In the original study, the authors use a batch size equivalent to 63 min of audio, and go through 400 000 updates. However, due to resource constraints and different data set up, a different strategy is used for these experiments. First, the batch size is chosen to be 512 as this is the largest value for which all models can be trained on a single Nvidia T4 GPUs. Secondly, pretraining is done for 200 epochs, i.e., the pretraining set is gone through 200 times, yielding a total of 26 580 updates. While the Data2Vec authors anneal the EMA decay over $n_\tau = 30\,000$ updates, this value is reduced to 1 000 to accommodate for the reduced number of updates. For masking, approximately 65 % of the input patch embeddings are masked by randomly sampling non-overlapping spans of $N_{\text{mask}} = 10$ patch embedding time steps. An average of the top $K = 8$ transformer layer output representations are used to form training targets. The final training target is obtained by applying instance normalization to the averaged representations of the transformer output representations. Optimization is done using a simple MSE loss

as the objective function and using the Adam optimizer [24]. The learning rate follows a 1-Cycle learning rate scheduler [54], where the learning rate is cosine annealed from a starting value $\eta_0 = \eta_{\max}/25$ to a maximum learning rate $\eta_{\max} = 500 \times 10^{-6}$, over the first 30 % of updates. The learning rate is then reduced over the remaining updates, also following a cosine annealing schedule. The 1-Cycle learning rate schedule is depicted in fig. 5.4. In addition to the learning rate scheduler, a weight decay of 0.01 is used to mitigate overfitting. These hyperparameter choices for Data2Vec pretraining are summarized in table 5.3.

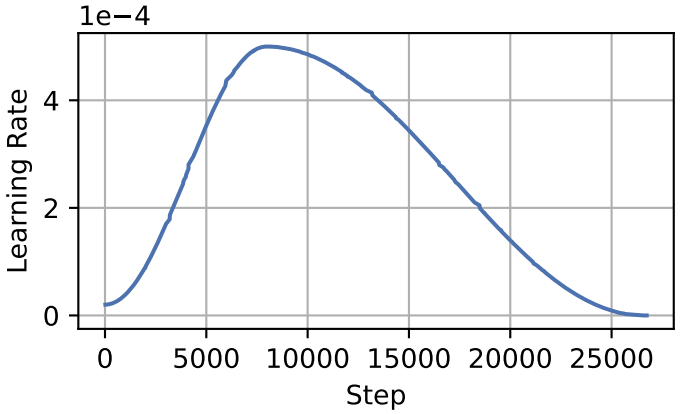


Figure 5.4: 1-Cycle

Table 5.3: Summary of settings used for Data2Vec pretraining.

τ_0	τ_{end}	n_τ	p_{mask}	N_{mask}	K
0.999	0.9999	1000	0.65	10	8
Epochs	Batch Size	Optimizer	Scheduler	Weight decay	Loss
200	512	Adam	1-cycle	0.1	MSE

It can be difficult to measure the pretraining performance of self-supervised models like Data2Vec using other measures than the loss. However, the loss converging to a low value does not necessarily mean that the network has learned good representations of the data, as is the case when model collapse happens. In order to get an idea of the richness in the learned representations, it is useful to look at the variance of the outputs from the teacher network (target) and student network (prediction). This can be used to spot model collapse, as the variance of the target and prediction would become very low. Additionally, the target and prediction variances can give a hint towards whether the model has converged, as the variances should also converge to a steady state. Nevertheless, the best way to test how good the learned representations are for KWS, is to fine-tune and evaluate the pretrained model on the reduced training set.

In fig. 5.5 the Data2Vec pretraining curves for all three KWT variations are seen. All models were trained for 200 epochs, with a batch size of 512 and using the hyperparameters seen in table 5.3. It should also be noted that the SpecAugment augmentation is not used for pretraining, and is thus only used for the baseline and fine-tuning of the KWT models. Due to instance normalization of the target, the target variance is always close to 1. The pretraining curves are very similar for all three KWT variations. As can be seen, the loss initially decreases significantly, while the prediction variance also decreases. At some point, the prediction

variance starts to increase towards the target variance, indicating that the model learns more rich representations. The prediction variance and loss are both seen to converge to a steady state value, whereafter they do not change much, indicating that the model does not further improve from training.

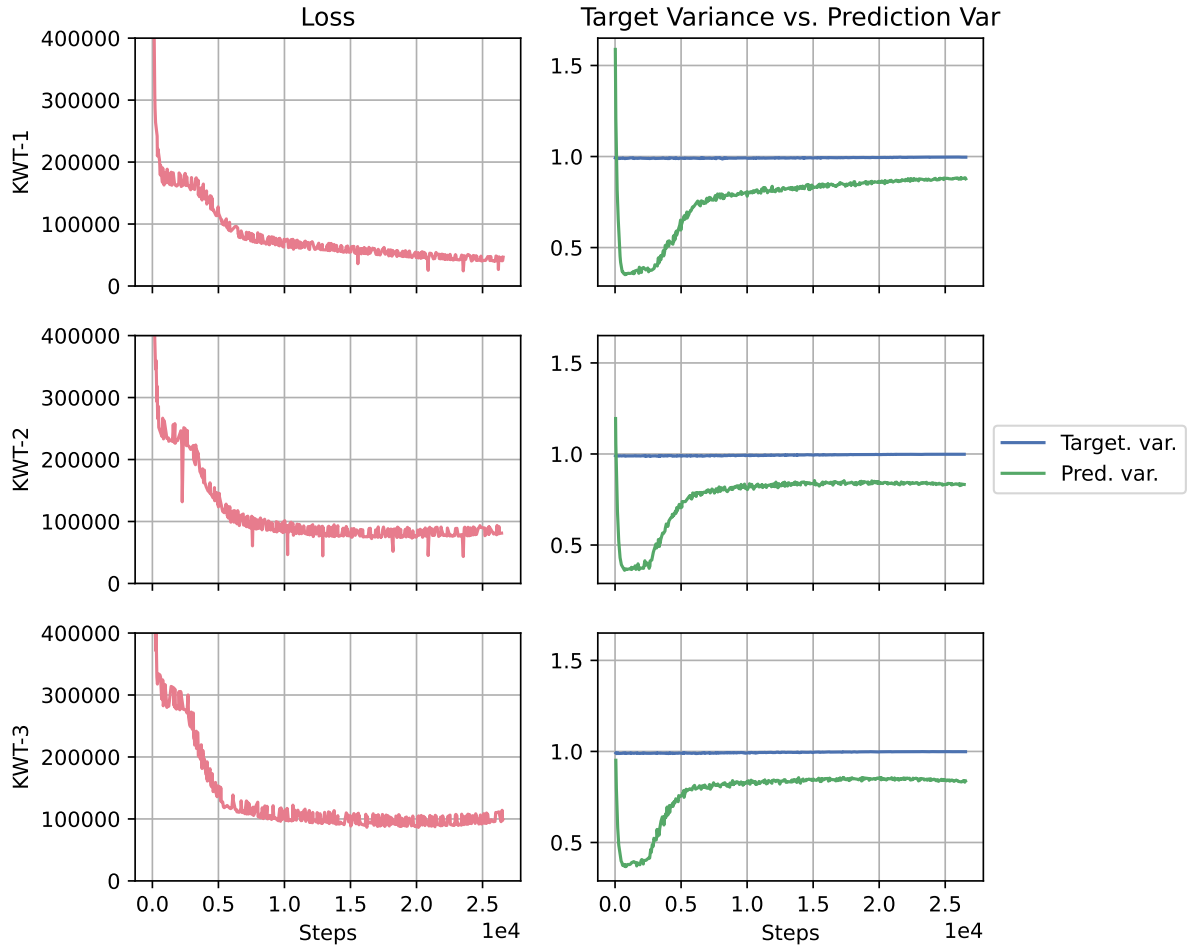


Figure 5.5: Pretraining curves for Data2Vec pretraining of KWT models.

5.4.2 Fine-tuning Pretrained Model

After pretraining the models on the pretraining data set using Data2Vec, they are fine-tuned on the reduced training set. Here, the prediction head used for Data2Vec pretraining is removed and the original classification head of the KWT model is used instead. The training paradigm is the same as that used for generating the baseline results. The resulting fine-tuning training curves are seen in fig. 5.6 together with the corresponding baseline training curves.

As can be seen from the training curves, the pretrained models achieve good performance in much fewer epochs than the baseline models. Moreover, they converge to a much higher accuracy than the baseline without pretraining. This is also depicted in the test set scores, which shows improvements in accuracy score ranging from 15.9 % to 18.87 % as seen in table 5.4.

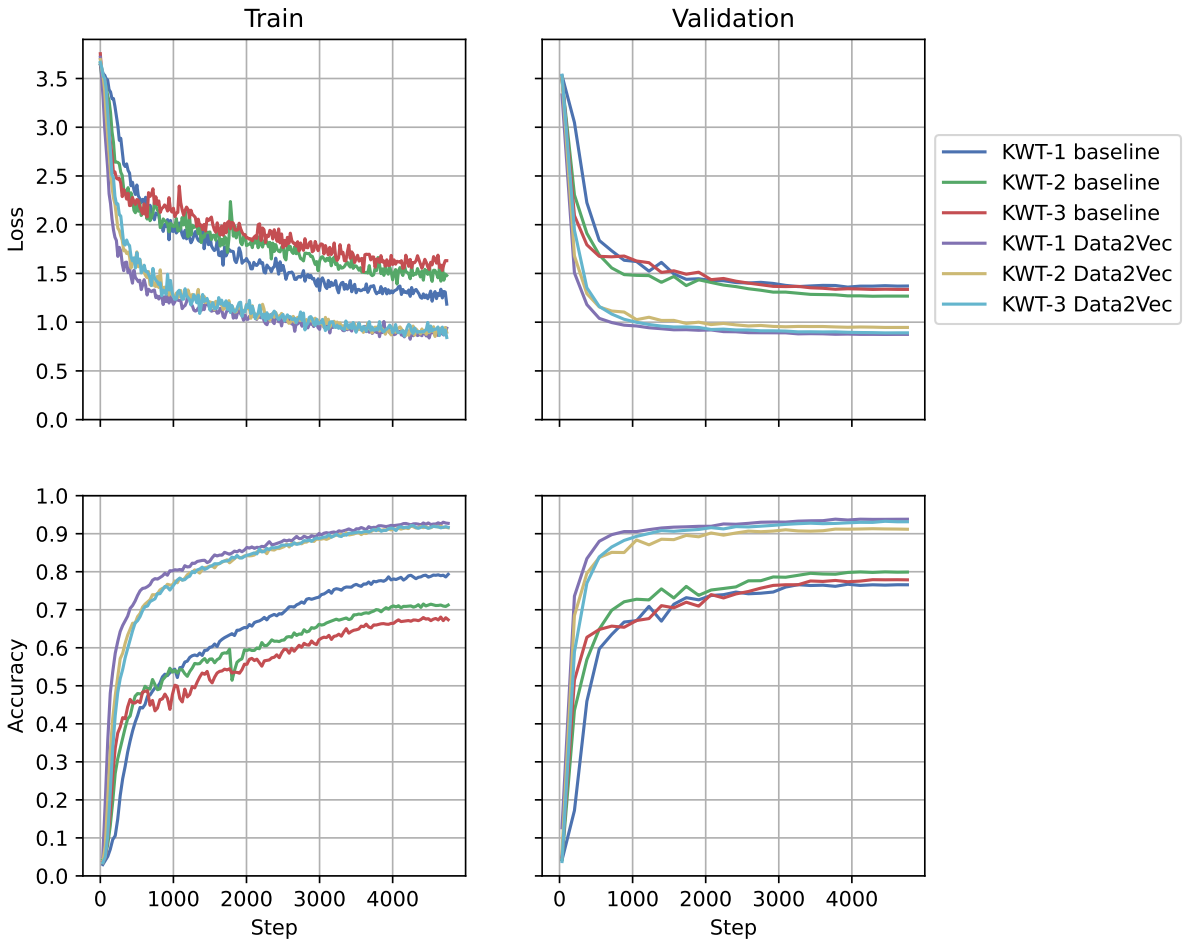


Figure 5.6: Training curves for fine-tuning of KWT models pretrained using Data2Vec compared to baselines.

Table 5.4: Summary of results for different KWT variations when pretrained using Data2Vec.

Model	Test Accuracy	Improvement over baseline
KWT1-data2vec	0.9286	0.1856
KWT2-data2vec	0.9448	0.1590
KWT3-data2vec	0.9465	0.1887

5.4.3 Ablation studies

The initial results show that Data2Vec pretraining on unlabelled data can significantly improve the performance of the KWT model when only a small amount of labelled data is available. However, it is also of interest to know how the performance changes, when varying the hyperparameters of the Data2Vec framework. Therefore, four different variations, in the Data2Vec pretraining setup has been tested, which are summarized in table 5.5. The base setup is named B and the variations are named V1, V2, V3 and V4.

In the first variation, V1, the instance normalization is replaced by layer normalization, keeping the other settings fixed. In the Data2Vec study, layer normalization is used for vision and text pretraining, however, as the KWT is basically a vision transformer that takes spectrogram patches as input instead of image patches, layer normalization might be equally

Table 5.5: Summary of different variations of the Data2Vec pretraining settings used for ablations studies.

Setting	Mask length	Mask probability [%]	Normalization type
B	10	65	Instance
V1	10	65	Layer
V2	10	20	Layer
V3	10	20	Instance
V4	5	65	Instance

applicable. The second variation, V2, additionally reduces p_{mask} from 65 % to 20 % in order to test the influence of the amount of masking applied. Reducing the amount of masked time steps should yield an easier prediction task, however, it might also mean that the learned representations become less rich and therefore lead to worse fine-tuning performance. The third variation, V3, also tests reducing the amount of masking when using the original instance normalization. Finally, V4 uses a smaller span of masked time steps by reducing the mask length from 10 time steps to 5 time steps, keeping a mask probability of 65 %. This is done to test the influence of the mask length. For audio, a smaller mask length should make the prediction task easier, as consecutive time step patches of the spectrogram are correlated The resulting Data2Vec pretraining curves of the four variations compared to the base pretraining settings are seen in fig. 5.7.

From fig. 5.7, it can be seen that the two layer normalized versions, V1 and V2, have distinctively different pretraining variance curves. Instead of the target having a constant variance of 1, it is initially some low value, which increases over time, with the student variance following. This can be seen as the teacher and student both gradually learning better representations. Additionally, it can be seen that the loss for V1 and V2 is significantly lower than the other variations, however this does not necessarily mean they perform better for fine-tuning.

While some distinctive differences can be seen in the pretraining curves for the different variations, it is more interesting to see how the different changes affect the fine-tuning performance. In fig. 5.8, the training curves when fine-tuning the different variations compared to the results using the base settings are seen, and their results on the test set are seen in table 5.6. Here it is seen that all the variations achieve a performance boost compared to the baseline model without pretraining, although, not all variations achieve as good performance as when using the base pretraining settings.

While varying the pretraining settings does not affect performance much in most cases, it is seen in table 5.6 that V1 and V2 yield significantly worse performance for the KWT-3 model. As can be seen in the training curve they both initially follow a similar pattern as the base settings, however at one point both of them suddenly decrease in performance and fall somewhere between the baseline model and the other pretrained models. The sudden decrease in accuracy makes it seem like the model forgets most of its learned representations, which is a phenomenon known as catastrophic forgetting [55]. Interestingly, this happens just before the learning rate reaches its highest value. This can indicate that the learning rate caused a big step in a 'bad' direction when updating the weights, however, further experiments would need to be carried out to verify this hypothesis.

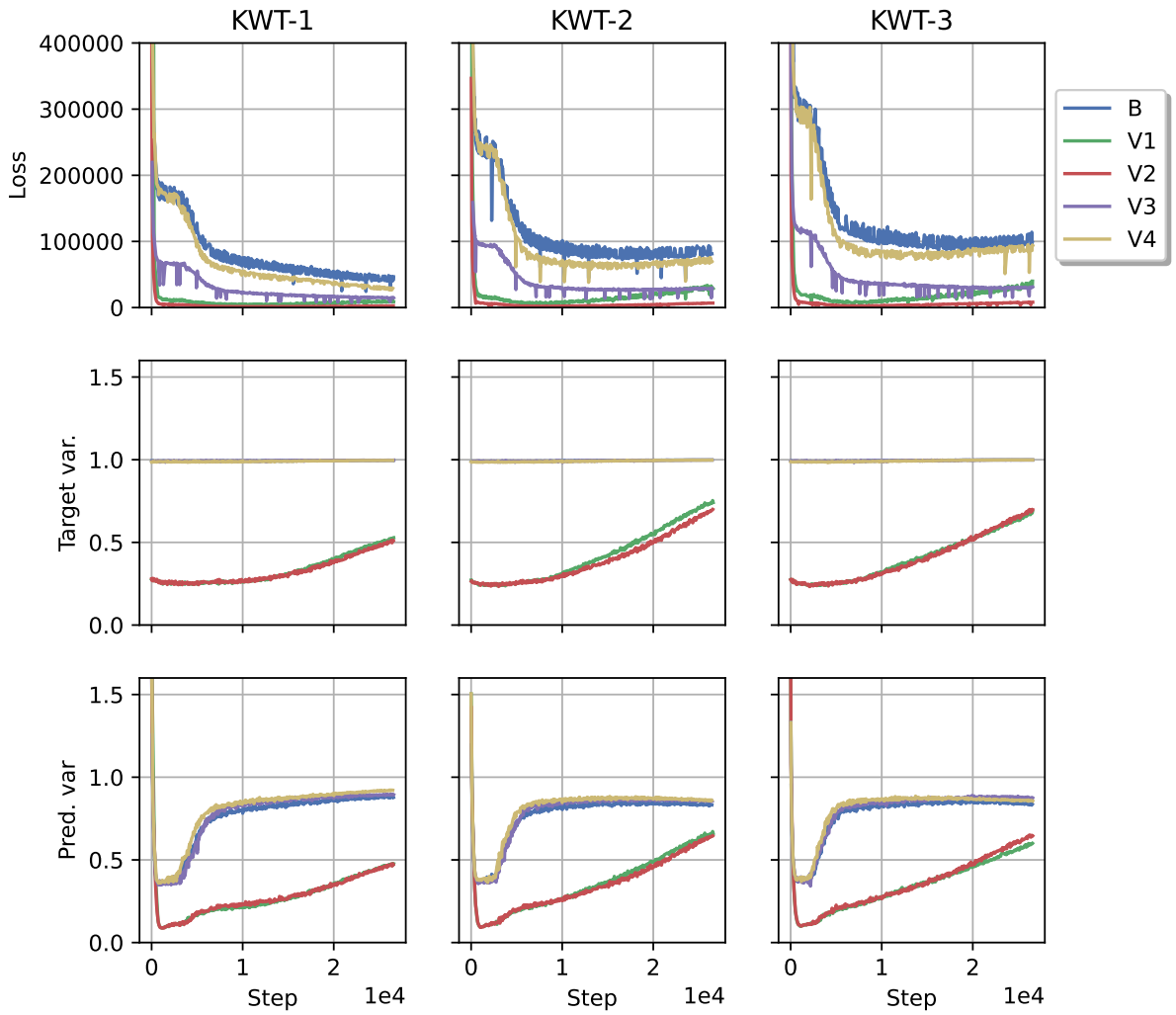


Figure 5.7: Comparison of pretraining curves when varying Data2Vec pretraining settings according to table 5.5.

5.4.4 Experiments Summary

From the experiments, it is clear that all three variations of the KWT model benefits from Data2Vec pretraining in the reduced labelled data setup. Additionally, in the ablation experiments, Data2Vec pretraining showed to perform almost equally well for all choices of pretraining hyperparameters. A significant performance drop compared to the base Data2Vec settings were seen when fine-tuning the KWT-3 model after pretraining using the V1 and V2 settings, with the KWT-3 model only achieving marginal improvements in accuracy in both cases. Common for both V1 and V2 is that they use layer normalization instead of instance normalization, however, the same pattern were not seen for KWT-1 and KWT-2.

The best accuracy score was achieved for the KWT-2 model when using the V4 settings, where an accuracy of 94.91 % was achieved. However, for the other choices of Data2Vec settings, the KWT-2 model generally achieved similar accuracies. In addition, the KWT-1 model performed best when pretrained using the V3 settings, and the KWT-3 model performed best using the base settings, thus the results do not point towards some 'best' choice of hyperparameters, but rather towards the fact that many choices of settings can lead to similar results.

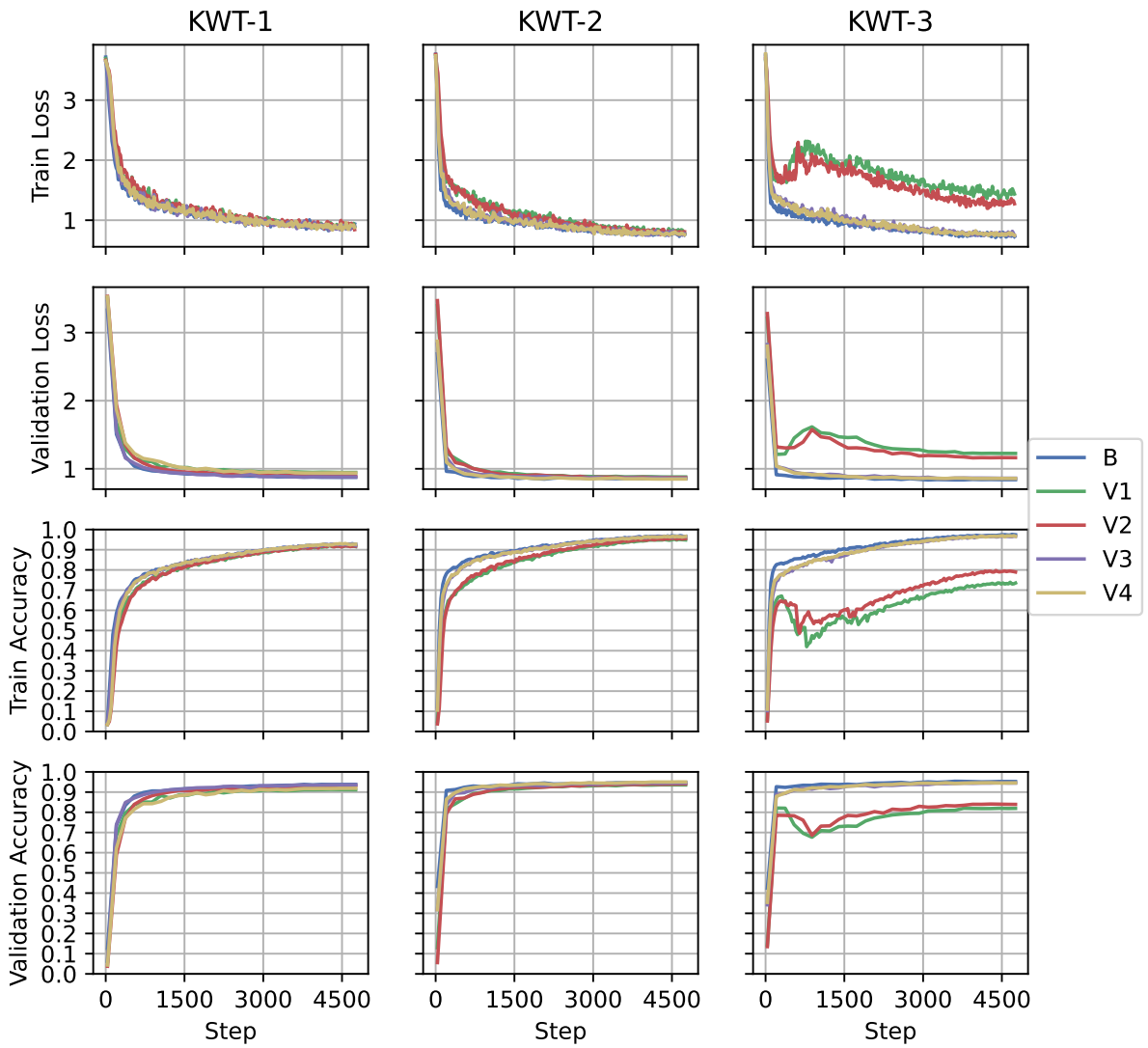


Figure 5.8: Training curves for fine-tuning KWT models when varying Data2Vec pretraining settings according to table 5.5

Generally, the use of Data2Vec pretraining significantly improved the performance of the KWT models on the reduced labelled setup of the Google Speech Commands V2 data set. Besides the KWT-3 pretrained with the V1 and V2 settings, all models achieved a performance boost in range of 15.56 % to 18.87 %.

Interestingly, model collapse was not experienced at any point, even when the hyperparameters were changed from those used in the original study. Additionally, a different model and different input data was used. This indicates that the measures used to prevent model collapse, namely target normalization and the use of EMA updates for the teacher model weight updates, sufficiently prevents model collapse from occurring, without needing to carefully tune hyperparameters to match the specific choice of model and input data. The fact that Data2Vec pretraining works rather well for different choices of hyperparameters, could also indicate that it can achieve good performance for a wide range of settings, although a more extensive study on the influence of hyperparameter choices would have to be carried out to verify this hypothesis.

Table 5.6: Test set accuracy scores for different settings of Data2Vec pretraining. B denotes base settings, V1 denotes Variation 1, V2 denotes Variation 2, V3 denotes Variation 3 and V4 denotes Variation 4

Model	Settings	Test Accuracy
KWT-1	B	0.9294
	V1	0.8986
	V2	0.9189
	V3	0.9308
	V4	0.9122
KWT-2	B	0.9449
	V1	0.9352
	V2	0.9386
	V3	0.9398
	V4	0.9491
KWT-3	B	0.9464
	V1	0.7969
	V2	0.8229
	V3	0.9357
	V4	0.9409

Chapter

6 Conclusion

This thesis aimed to investigate the use of the self-supervised learning framework Data2Vec for improving the performance of keyword spotting systems when only sparse amounts of labelled data is available. This was inspired by the fact that many KWS applications are limited to languages with large available speech databases due to the fact that the current supervised methods need large amounts of labelled data. Initially, the subject of self-supervised learning and how it can be used for deep KWS in sparse labelled setups was presented, along with a description of a recently published general self-supervised framework called Data2Vec. It was identified that the Data2Vec framework had only been applied to very large models with several million parameters, which raised the question if the Data2Vec framework can also be used for pretraining of deep KWS models which are much smaller. As a result, it was decided to investigate whether Data2Vec pretraining can increase the performance of KWS models, when only a small amount of labelled data is available. The following research question was then formulated:

> Can Data2Vec pretraining using unlabelled data improve the performance of keyword spotting models, when only a small amount of labelled data is available?

Following the above research question, an introduction to the current state of KWS and the various parts of a KWS system was first presented. A general overview of the area of self-supervised learning was then presented, along with a description of the newly proposed Data2Vec framework. The implementation of a self-supervised KWS system using Data2Vec pretraining was then described. Here, it was chosen to use the transformer based KWTmodel, following the use of a transformer encoder in the original Data2Vec study. Three variations of the KWTmodel were implemented, namely KWT-1, KWT-2 and KWT-3, each model increasing in size, ranging from 607×10^3 to $5\,361 \times 10^3$ parameters. This was done to be able to test the influence of model size on Data2Vec pretraining performance. The implemented system was then tested on a reduced labelled setup of the Google Speech Commands V2 data set, created to emulate a sparse labelled data setup.

In the experiments, a baseline for the KWTmodel without pretraining was first established for the reduced labelled setup. The best performing of the three KWT variations was the

KWT-2 model, which achieved a test set accuracy of 78.69 %, with the others scoring a few percent point lower. With the baseline results established, the KWT models were pretrained on a set aside pretraining set using Data2Vec, after which they were fine-tuned following the same training procedure used to generate the baseline results. Here, all three models showed significant improvements in accuracy. The best improvement was achieved for KWT-3 with an improvement of 18.56 %, while the accuracy of KWT-1 and KWT-2 improved by 15.9 % and 18.87 %, respectively. Based on these results, it can be concluded that Data2Vec successfully increases the performance of the implemented KWS system by a relatively large margin compared to the baseline results without pretraining. Ablation studies furthermore showed that the performance boost from Data2Vec pretraining was consistent for multiple choices of Data2Vec pretraining hyperparameters, such as varying amounts of masking. The results show that Data2Vec can successfully be used for relatively small KWS models, and is thus not reserved for large models with tens of millions of parameters. Additionally, the results show that Data2Vec can significantly improve KWS performance when only a small amount of labelled data is available.

While the results show that Data2Vec pretraining can be used to improve KWS performance in sparse labelled data setups, one question is whether these results translate to other choices of model and different data setups. Further research is also needed to determine how Data2Vec performs in noisy conditions, as the data used in this study only included clean keyword segments. The choice of hyperparameter settings for the Data2Vec method are shown not affect fine-tuning performance much, however, more extensive efforts to determine the influence of Data2Vec hyperparameters are needed, in order to be able to make a general conclusion on these findings.

In this thesis, experiments were carried out using a set aside pretraining dataset with data similar to the training data. Alternatively, it would be interesting to replace the pretraining set with a more general speech data set such as Librispeech, to test whether this can further improve the learned representations. Using a more general pretraining set could also open up for a more general study on the applicability of learned representations for multiple use cases, other than KWS. Future studies might also address the general applicability of Data2Vec for additional use cases where smaller models are desired.

Although, the Data2Vec framework has currently only been applied to transformer architectures, it might be equally applicable to other architectures such as CNN-based or RNN-based models. Therefore, investigation into whether Data2Vec can be adapted to other architectures is also an interesting direction, which can potentially generalize the use of the Data2Vec framework to a broader range of models, making it more generally applicable.

To summarize, it has been shown that pretraining using Data2Vec can significantly improve KWS performance when only a small amount of labelled data is available. Moreover, the Data2Vec method showed to be relatively invariant to the choice of method specific pretraining hyperparameters. While these results are promising, more in-depth research into the impact of model choice and data set have to be carried out, in order to generalize these findings.

Bibliography

[1] Y. LeCun and I. Misra. 'Self-supervised learning: The dark matter of intelligence.' (2021), [Online]. Available: https://ai.facebook.com/blog/self-supervised-learning-the-dark-matter-of-intelligence/ (visited on 18/05/2022).

[2] S. Schneider, A. Baevski, R. Collobert and M. Auli, 'Wav2vec: Unsupervised pre-training for speech recognition,' *CoRR*, vol. abs/1904.05862, 2019. [Online]. Available: http://arxiv.org/abs/1904.05862.

[3] M. L. Anderson, 'Neural reuse: A fundamental organizational principle of the brain,' *Behavioral and Brain Sciences*, vol. 33, no. 4, pp. 245–266, 2010. DOI: 10.1017/S0140525X10000853.

[4] I. López-Espejo, Z. Tan, J. H. L. Hansen and J. Jensen, 'Deep spoken keyword spotting: An overview,' *CoRR*, vol. abs/2111.10592, 2021. [Online]. Available: https://arxiv.org/abs/2111.10592.

[5] Y. Zhuang, X. Chang, Y. Qian and K. Yu, 'Unrestricted vocabulary keyword spotting using lstm-ctc,' Sep. 2016, pp. 938–942. DOI: 10.21437/Interspeech.2016-753.

[6] A. Bardes, J. Ponce and Y. LeCun, 'Vicreg: Variance-invariance-covariance regularization for self-supervised learning,' *CoRR*, vol. abs/2105.04906, 2021. [Online]. Available: https://arxiv.org/abs/2105.04906.

[7] J. Grill, F. Strub, F. Altché *et al.*, 'Bootstrap your own latent: A new approach to self-supervised learning,' *CoRR*, vol. abs/2006.07733, 2020. [Online]. Available: http://arxiv.org/abs/2006.07733.

[8] A. Baevski, W. Hsu, Q. Xu, A. Babu, J. Gu and M. Auli, 'Data2vec: A general framework for self-supervised learning in speech, vision and language,' *CoRR*, vol. abs/2202.03555, 2022. [Online]. Available: https://arxiv.org/abs/2202.03555.

[9] K. Friston and S. Kiebel, 'Predictive coding under the free-energy principle,' *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 364, no. 1521, pp. 1211–1221, 2009. DOI: 10.1098/rstb.2008.0300.

[10] A. Jaegle, F. Gimeno, A. Brock, A. Zisserman, O. Vinyals and J. Carreira, 'Perceiver: General perception with iterative attention,' *CoRR*, vol. abs/2103.03206, 2021. [Online]. Available: https://arxiv.org/abs/2103.03206.

[11] Open SLR. 'Librispeech asr corpus.' (2022), [Online]. Available: https://www.openslr.org/12 (visited on 24/05/2022).

[12] A. Viterbi, 'Error bounds for convolutional codes and an asymptotically optimum decoding algorithm,' *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967. DOI: 10.1109/TIT.1967.1054010.

[13] M. Sun, D. Snyder, Y. Gao *et al.*, 'Compressed time delay neural network for small-footprint keyword spotting,' in *Interspeech 2017*, 2017. [Online]. Available: https://www.amazon.science/publications/compressed-time-delay-neural-network-for-small-footprint-keyword-spotting.

[14] I. López-Espejo, Z.-H. Tan and J. Jensen, *Exploring filterbank learning for keyword spotting*, 2020.

[15] R. C. Staudemeyer and E. R. Morris, 'Understanding LSTM - a tutorial into long short-term memory recurrent neural networks,' *CoRR*, vol. abs/1909.09586, 2019. [Online]. Available: http://arxiv.org/abs/1909.09586.

[16] J. Chung, Ç. Gülçehre, K. Cho and Y. Bengio, 'Empirical evaluation of gated recurrent neural networks on sequence modeling,' *CoRR*, vol. abs/1412.3555, 2014. [Online]. Available: http://arxiv.org/abs/1412.3555.

[17] Y. Yuan, Z. Lv, S. Huang and L. Xie, 'Verifying deep keyword spotting detection with acoustic word embeddings,' in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2019, pp. 613–620. DOI: 10.1109/ASRU46091.2019.9003781.

[18] N. Sacchi, A. Nanchen, M. Jaggi and M. Cernak, 'Open-vocabulary keyword spotting with audio and text embeddings,' in *INTERSPEECH*, 2019.

[19] K. He, X. Zhang, S. Ren and J. Sun, 'Deep residual learning for image recognition,' *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385.

[20] A. Vaswani, N. Shazeer, N. Parmar *et al.*, 'Attention is all you need,' *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762.

[21] A. Dosovitskiy, L. Beyer, A. Kolesnikov *et al.*, 'An image is worth 16x16 words: Transformers for image recognition at scale,' *CoRR*, vol. abs/2010.11929, 2020. [Online]. Available: https://arxiv.org/abs/2010.11929.

[22] A. Berg, M. O'Connor and M. T. Cruz, 'Keyword transformer: A self-attention model for keyword spotting,' in *Interspeech 2021*, ISCA, Aug. 2021. DOI: 10.21437/interspeech.2021-1286. [Online]. Available: https://doi.org/10.21437%5C%2Finterspeech.2021-1286.

[23] L. Liu, J. Liu and J. Han, 'Multi-head or single-head? an empirical comparison for transformer training,' *CoRR*, vol. abs/2106.09650, 2021. [Online]. Available: https://arxiv.org/abs/2106.09650.

[24] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017.

[25] P. Warden, 'Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition,' *ArXiv e-prints*, Apr. 2018. [Online]. Available: https://arxiv.org/abs/1804.03209.

[26] W. Commons. 'Receiver operating curve.' File: Roc curve.svg. (2018), [Online]. Available: https://en.wikipedia.org/wiki/File:Roc_curve.svg#metadata.

[27] L. Jing, P. Vincent, Y. LeCun and Y. Tian, 'Understanding dimensional collapse in contrastive self-supervised learning,' *CoRR*, vol. abs/2110.09348, 2021. [Online]. Available: https://arxiv.org/abs/2110.09348.

[28] X. Chen, H. Fan, R. B. Girshick and K. He, 'Improved baselines with momentum contrastive learning,' *CoRR*, vol. abs/2003.04297, 2020. [Online]. Available: https://arxiv.org/abs/2003.04297.

[29] A. Jaiswal, A. R. Babu, M. Z. Zadeh, D. Banerjee and F. Makedon, 'A survey on contrastive self-supervised learning,' *CoRR*, vol. abs/2011.00362, 2020. [Online]. Available: https://arxiv.org/abs/2011.00362.

[30] T. Chen, S. Kornblith, M. Norouzi and G. E. Hinton, 'A simple framework for contrastive learning of visual representations,' *CoRR*, vol. abs/2002.05709, 2020. [Online]. Available: https://arxiv.org/abs/2002.05709.

[31] D. Jiang, W. Li, M. Cao *et al.*, 'Speech SIMCLR: combining contrastive and reconstruction objective for self-supervised speech representation learning,' *CoRR*, vol. abs/2010.13991, 2020. [Online]. Available: https://arxiv.org/abs/2010.13991.

[32] H. Al-Tahan and Y. Mohsenzadeh, 'CLAR: contrastive learning of auditory representations,' *CoRR*, vol. abs/2010.09542, 2020. [Online]. Available: https://arxiv.org/abs/2010.09542.

[33] A. K. Sarkar and Z. Tan, 'Time-contrastive learning based unsupervised DNN feature extraction for speaker verification,' *CoRR*, vol. abs/1704.02373, 2017. [Online]. Available: http://arxiv.org/abs/1704.02373.

[34] X. Chen and K. He, 'Exploring simple siamese representation learning,' *CoRR*, vol. abs/2011.10566, 2020. [Online]. Available: https://arxiv.org/abs/2011.10566.

[35] D. Niizumi, D. Takeuchi, Y. Ohishi, N. Harada and K. Kashino, *Byol for audio: Self-supervised learning for general-purpose audio representation*. DOI: 10.48550/ARXIV.2103.06695. [Online]. Available: https://arxiv.org/abs/2103.06695.

[36] Y. Chung, W. Hsu, H. Tang and J. R. Glass, 'An unsupervised autoregressive model for speech representation learning,' *CoRR*, vol. abs/1904.03240, 2019. [Online]. Available: http://arxiv.org/abs/1904.03240.

[37] Facebook AI. 'Libri-light.' (2022), [Online]. Available: https://ai.facebook.com/tools/libri-light/ (visited on 24/05/2022).

[38] W. Hsu, B. Bolte, Y. H. Tsai, K. Lakhotia, R. Salakhutdinov and A. Mohamed, 'Hubert: Self-supervised speech representation learning by masked prediction of hidden units,' *CoRR*, vol. abs/2106.07447, 2021. [Online]. Available: https://arxiv.org/abs/2106.07447.

[39] PyTorch. 'An open source machine learning framework.' (2021), [Online]. Available: https://pytorch.org/ (visited on 09/05/2022).

[40] M. Morshed. 'Torch kwt.' (2021), [Online]. Available: https://github.com/ID56/Torch-KWT (visited on 24/05/2022).

[41] Facebook AI. 'Fairseq.' (2022), [Online]. Available: https://github.com/facebookresearch/fairseq (visited on 24/05/2022).

[42] B. McFee, C. Raffel, D. Liang *et al.*, 'Librosa: Audio and music signal analysis in python,' Jan. 2015, pp. 18–24. DOI: 10.25080/Majora-7b98e3ed-003.

[43] J. Devlin, M. Chang, K. Lee and K. Toutanova, 'BERT: pre-training of deep bidirectional transformers for language understanding,' *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805.

[44] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles and H. Jégou, 'Training data-efficient image transformers & distillation through attention,' *CoRR*, vol. abs/2012.12877, 2020. [Online]. Available: https://arxiv.org/abs/2012.12877.

[45] Hugging Face. 'The ai community building the future.' (2022), [Online]. Available: https://huggingface.co/ (visited on 24/05/2022).

[46] NVIDIA. 'Cuda toolkit.' (2022), [Online]. Available: https://developer.nvidia.com/cuda-toolkit (visited on 24/05/2022).

[47] CLAUUDIA, Aalborg University. 'Ai cloud.' (2022), [Online]. Available: https://www.claaudia.aau.dk/platforms-tools/compute/gpu-cloud-ai/ (visited on 24/05/2022).

[48] ——, 'Strato compute cloud.' (2022), [Online]. Available: https://www.claaudia.aau.dk/platforms-tools/compute/compute-cloud/ (visited on 24/05/2022).

[49] D. S. Park, W. Chan, Y. Zhang *et al.*, 'SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition,' in *Proc. Interspeech 2019*, 2019, pp. 2613–2617. DOI: 10.21437/Interspeech.2019-2680.

[50] I. Loshchilov and F. Hutter, 'Fixing weight decay regularization in adam,' *CoRR*, vol. abs/1711.05101, 2017. [Online]. Available: http://arxiv.org/abs/1711.05101.

[51] ——, 'SGDR: stochastic gradient descent with restarts,' *CoRR*, vol. abs/1608.03983, 2016. [Online]. Available: http://arxiv.org/abs/1608.03983.

[52] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, 1st ed. MIT Press, 2017, ISBN: 978-0262-03561-3.

[53] R. Müller, S. Kornblith and G. E. Hinton, 'When does label smoothing help?' *CoRR*, vol. abs/1906.02629, 2019. [Online]. Available: http://arxiv.org/abs/1906.02629.

[54] L. N. Smith and N. Topin, 'Super-convergence: Very fast training of residual networks using large learning rates,' *CoRR*, vol. abs/1708.07120, 2017. [Online]. Available: http://arxiv.org/abs/1708.07120.

[55] P. Kaushik, A. Gain, A. Kortylewski and A. L. Yuille, 'Understanding catastrophic forgetting and remembering in continual learning with optimal relevance mapping,' *CoRR*, vol. abs/2102.11343, 2021. [Online]. Available: https://arxiv.org/abs/2102.11343.