Robust Keyword Spotting Using Self-Supervised Deep Learning

Master Thesis

Gergely Kiss / Group 972

Aalborg University
Electronics and IT

Electronics and IT
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Robust Keyword Spotting Using Self-Supervised Deep Learning

Theme:
SPA10 Long Master Thesis

Project Period:
September 2022 - June 2023

Project Group:
Group 972

Participant(s):
Gergely Kiss

Supervisor(s):
Zheng-Hua Tan
Holger Severin Bovbjerg

Copies: 1

Page Numbers: 55

Date of Completion:
June 2, 2023

Abstract:

Keyword spotting (KWS) is a subtask of automatic speech recognition focusing on recognizing a limited number of words in the audio stream. KWS systems must meet the demand of ever-decreasing complexity, increasing accuracy, and robustness against acoustic noise. Many methods have been developed to increase the latter. The robustness of a novel KWS model called Keyword Transformer (KWT) has not yet been studied. It has been shown that self-supervised pretraining of KWT in a data2vec system can enhance the accuracy of keyword spotting. This study aims to provide supervised and self-supervised training methods to increase the robustness of the KWT, namely multi-style and adversarial learning. The models were trained and tested on the reduced Google Speech Commands Dataset and noise was added from CHiME-3. Multi-style supervised training increased the model accuracy significantly in noisy conditions, while adversarial training showed only marginal enhancement. Moreover, self-supervised pretraining on clean speech has shown increased robustness with a combination of both.

Contents

Preface

This is the report of the Long Master Thesis project of Gergely Kiss in Signal Processing and Computing at Aalborg University.

Reading Guide

The author of the report assumes that the reader has a basic understanding of machine learning and neural network concepts such as artificial neurons, activation functions, loss functions, or common types of neural networks.

The report follows the IEEE format. Fractional values are divided with decimal points. Figures are made using the Draw.io graphical software and the Matplotlib Pyplot and Librosa Pythong libraries.

The thesis project is the continuation of the project Self-Supervised Keyword Spotting Using Data2vec by Holger Severin Bovbjerg [5]. Therefore, some of the information is not explained in detail, only the relevant to this project.

The report is structured as follows: In the introduction, chapter 1, the main problem is explained briefly, and the research question is presented. In chapter 2, the keyword spotting and the robustness of the keyword spotting are shown. In chapter 3, the neural networks that serve as the basis of this work are expounded. Chapter 4 presents other studies about enhancing the robustness of speech recognition systems that served as the inspiration for this project. In chapter 5, methods to improve the Keyword Transformer are explained. Chapter 6 presents the methods and results of the experiments. Chapter 7 concludes the work.

Acknowledgements

I would like to thank Zheng-Hua Tan for his guidance and insightful advice. His deep understanding and clear explanations helped me a lot during the project. I also would like to thank Holger Severin Bovbjerg, whose thorough insights into the Keywords Transformers and data2vec and practical and technical advice were helpful during implementation and problem-solving.

Aalborg University, June 2, 2023

Gergely Kiss
<gkiss21@student.aau.dk>

Chapter 1

Introduction

In recent decades, deep neural networks (DNNs) have become the machine learning paradigm in several signal-processing domains, replacing other probabilistic machine learning methods such as Support Vector Machines or Gaussian mixture modes. Artificial neural networks are mathematical models of the synaptic biological neural network of the human brain.

Different kinds of deep neural networks have been widely used in various modalities for various tasks with great success. These models have been specially designed for specific tasks; their structures, sizes, or feature extraction methods are firmly task and modality-specific. For example, convolutional neural networks excel at image classification or recurrent neural networks for sequential data generation. In recent years there has been a breakout in research for modality-agnostic models based on the theory that humans use the same part of the brain the learn different tasks, such as vision or hearing.

At first, neural network models were trained in supervised training on labeled datasets. With time the demand for larger labeled datasets increased, leading to another challenge: creating labeled datasets required human labor, which was time-consuming and expensive. Such datasets had thousands or millions of elements, and each of them had to be labeled by humans putting a solid limit on the size of the dataset and, thus, the training and final capability of the networks.

One solution to this problem was transfer learning. The idea behind this method is that a neural network can perform better during training if its parameters are initiated in a way that it already has some knowledge about the features of the training dataset, giving a good start to the learning. For example, a convolutional neural network trained for recognizing animals can be finetuned to identify plants. Although animals might differ from plants, some visual features, like edges or other shapes, can be similar, leading to a domain shift on the feature level. In this example, transfer learning is supervised learning both for animal and plant recognition. Although it might work well for a reduced plant dataset, it still requires a reasonably large labeled dataset for the animal classification task. Another problem with transfer learning is that it is heavily dependent on the source task, and a cautious selection of it is crucial.

While supervised learning required labels to form the learning targets, unsupervised learning did not. The purpose of unsupervised learning was not to connect specific data with a label but to cluster the dataset. Although it shows great potential in many applications, its usage is limited. Self-supervised learning, another learning method, forms a target from the data itself. A target can be continuous for regression tasks like a masked prediction [1] or discrete for classification tasks like image rotation prediction [51]. Self-supervised learning is usually used as a pretext task for

pretraining neural networks. The pretext task aims to learn a good representation of the dataset, such as visual tokens for the image task or linguistic tokens for the natural language processing (NLP) task. The main advantage of self-supervised learning is that it can be trained on a large unlabeled dataset. While labeled datasets are sparse and small, unlabeled datasets are widely available on the Internet, and their size is virtually infinite. Self-supervised pretraining has shown great potential to increase the accuracy of neural nets.

Data2vec [1] is a recently published self-supervised modality-independent model that has set state-of-the-art or competitive performance in natural language processing, speech processing, and computer vision, showing the strength of self-supervised pretraining. Data2vec trains a transformer network in a student-teacher model, where the student gets a masked version, and the teacher receives the unmasked version of the input data. The student predicts the continuous and contextualized latent outputs of the transformer layers of the teacher. The teacher's weights are updated as the exponentially decaying average of the student's weights. This pretraining method was shown to be more effective than previous works' task-specific approaches, while data2vec's pretraining is the same in all modalities.

One of the tasks in which self-supervised pretraining has shown a remarkable increase in performance is automatic speech recognition (ASR). Historically ASR algorithms used hidden Markov models (HMMs) [52] or dynamic time warping (DTW) [13]. Nowadays, DNNs are used as the main method for speech recognition for their better accuracy and other capabilities, such as recognizing dialects or moods. Unfortunately, running ASR algorithms is a computationally heavy task and requires high computational power computers. On the other hand, one subtask of speech recognition is specially designed to run on smaller devices with limited computational power: keyword spotting. The goal of keyword spotting algorithms is to recognize certain words in the sentence without understanding the whole speech. Such algorithms can be used for embedded devices such as trigger words (e.g., "OK Google," "Hey Siri") for voice assistants or limited voice commands for other systems [15].

Keyword spotting algorithms are usualy evaluated on the same domain they are trained on. However, this evaluation does not necessarily show the network's real practical performance when the prediction domain changes by changing the speaker's speech characteristics or some background noises are introduced. The robustness to acoustic variations of a DNN model refers to how well a neural network performs in such an unideal environment.

Many studies have been conducted on improving the robustness of ASR and speech enhancement (SE) algorithms. They bear significant practical importance, such as enhancing phone calls, hearing aids, or voice assistant devices. In practice, such algorithms must handle previously unknown noises in various signal-to-noise ratios (SNRs) while maintaining their core functionalities. At first statistical methods, such as the Gaussian mixture model (GMM) or support vector machine (SVM), were used; however, nowadays, DNN outperforms both approaches and has been the primary method for ASR or SE tasks. Improving the noise robustness of a neural network can be done on many levels, e.g., applying particular data augmentation before feeding the network or introducing new training approaches such as adversarial training [15].

1.1 Research Question

The robustness of different neural networks has been widely studied in speech recognition tasks, including keyword spotting. These studies bear critical importance in practical applications since the average use case of keyword spotting algorithms is in an environment where some level of noise

is present. Another requirement of the keyword spotting algorithm is to be lightweight for use on a mobile platform with low computational resources while producing low latency and consuming as little battery power as possible.

Keyword Transformer [4] is a Transformer-based [53] keyword spotting model that achieved state-of-the-art performance against larger models on Google Speech Commands V2 [55] dataset. In the study of Bovbjerg [6], the testing performance of the Keyword Transformer was enhanced by pretraining it in the data2vec system, which showed great opportunity in applications where the training dataset is sparse.

Increasing the noise robustness of the Keyword Transformer can allow practical applications to implement robust lightweight keyword spotting systems where only little labeled training data is available. The research question thus becomes as follows:

What supervised and self-supervised training methods can be used to enhance the noise-robustness of the Keyword Transformer?

Chapter 2

Keyword Spotting

In the last few years, automatic speech recognition algorithms have developed so far that they are now omnipresent in many devices. Such devices can expand human-machine interaction, enhance speech for hearing assistive devices, or be used for data mining. Automatic speech recognition algorithms convert human speech to text, where further data handling can be executed. One subproblem of ASR is keyword spotting. Keyword spotting is used to recognize the presence of a word or phrase (e.g., "Hey Siri") in the speech giving out posterior probabilities of whether a keyword is present and also a probability of which keyword was present.

First, keyword spotting algorithms used continuous speech recognition systems, which converted the whole speech to text and then found the keyword in the text. This application, however, required immense computation power and energy consumption. In an application like this, most of the analyses were useless since the effective use was to spot a limited number of words amongst a plethora of others. Later, keyword-spotting algorithms concentrated on recognizing only certain words without analyzing the whole speech leading to more lightweight algorithms. Such algorithms made spreading on embedded devices with limited computational resources possible, for example, mobile phones.

2.1 Practical applications

The keyword spotting system's most practical application is activating voice assistants. Being lightweight, these KWS algorithms can run on devices with limited computational resources, such as mobile phones or smart home devices. When the trigger word or phrase is said, the smart device sends the voice command to a client-server, where it is further processed, and the command is executed. These devices have to work on low false activation (FA) and false rejection (FR) rates. In cases like this, KWS algorithms have to be lightweight to meet the computational constraints and keep the power consumption as low as possible since, to catch the activation phrase, it must continuously monitor the sound through the microphone [15].

Other applications of KWS systems can be when a particular device is expected to be controlled by voice. These devices do not wake up more complex ASR systems but execute the command by themselves. Such applications can be considered to reduce the complexity of handling the device generally or make the device more accessible to disabled people. One example is that some devices can be difficult to handle by hand. Such devices can be hearing aids where it might be easier to

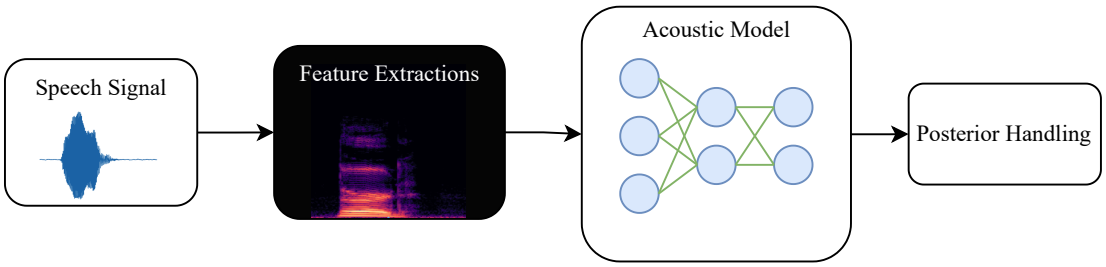

Figure 2.1: Main pipeline of a DNN-based keyword spotting system

raise the device's volume by giving voice commands. Such an application must be activated by the person who wears the device. Another application can be voice-controlled accessible pedestrian signals (APS). A push-button APS device can be difficult to be operated by a vision-impaired person. Other examples can be car applications such as controlling the air conditioning by voice. Using such an application, the driver is freed from using the control panel, which could divert his attention and increase the possibility of an accident. [15]

2.2 Deep Keyword Spotting Approach

Although, in time, many different techniques have been developed for keyword spotting, most of the application share a very similar structure for handling spoken keywords. Modern DNN-based keyword spotting systems usually have three main blocks:

1. Speech feature extractor
2. Acoustic model
3. Posterior handler

This model is the basis of modern algorithms. Different application designs these blocks differently, but the main structure is usually kept. In this section, these blocks are explained. Figure 2.1 shows the main pipeline of such as system [15].

2.2.1 Speech Feature Extractor

Selecting the proper feature extraction method is a crucial step in speech recognition. By doing so, the most representative features of the speech can be emphasized while suppressing unimportant or even undesired ones. Learning of speech is done by those extracted parameters. Thus, selecting features that best represent human speech characteristics is desired. Feature extraction has to be sufficiently fast if a real-time application is considered while minimizing the loss of useful information. Choosing a feature extraction method or its parameters heavily depends on the acoustic applications. Different settings can be used, for example, if the full word is considered the basis of a classification in contrast to if the classification is done on the phonetic level.

One of the most prominent feature extraction methods used for decades is based on the two-dimensional time-frequency spectrum of the audio input. These spectrograms are created by short-time Fourier or short-time cosine transforms with a sliding window. The size of the window and the

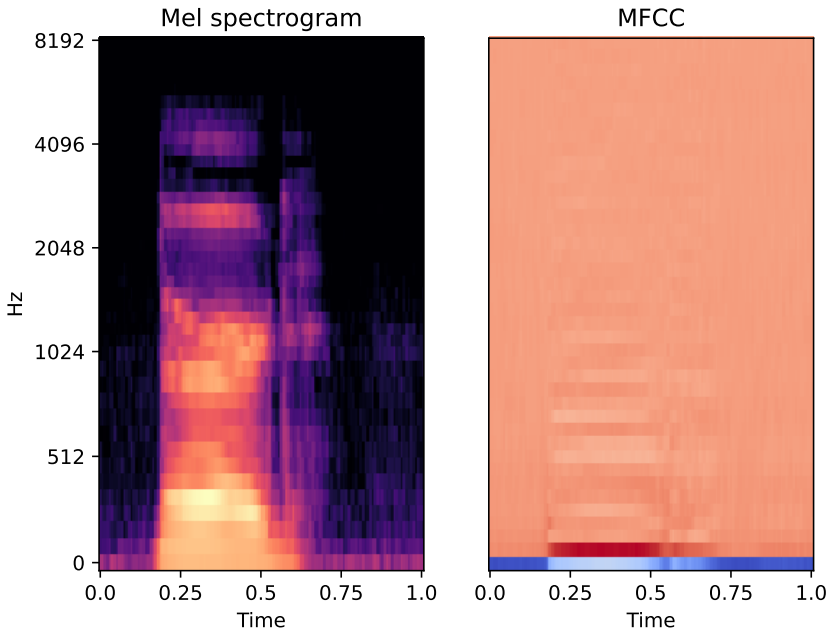

Figure 2.2: Mel-scale spectrogram and MFCC features of a person's voice saying "bird."

resolution of the cosine (or Fourier) transform defines the parameter of the spectrogram given input audio with a certain length. The most relevant feature extraction methods are the log-Mel spectral coefficients or the Mel-frequency cepstral coefficients (MFCCs). The notion behind these methods is the functionality of the human ear. The ear is inequally sensitive to the different frequency ranges, mostly linear below and logarithmic above 1000 Hz. The Mel-scale is based on such pitch perception, and Mel-scale filter banks created to extract the MFCCs are designed according to that [16]. Figure 2.2 shows an example of a mel-scale spectrogram and MFCCs. Mel-frequency cepstrum (MFCs) model the human auditory perception. Hence, mel-scale-based feature extractions are one of the most frequently used methods for ASR algorithms.

Another widely used feature extraction method is done by recurrent neural networks (RNNs). RNNs are effective in one-dimensional applications where the input is encoded as a vector. Thanks to their recursive structure, RNNs are capable of handling various lengths of input. Short-term memory recurrent neural networks are successfully used in several KWS applications with query-by-example (QbyE) keyword detection methods. QbyE takes several examples of keywords as a template and compares the test audio with it [8].

2.2.2 Acoustic Model

The acoustic model is the main element of deep KWS element. It is responsible for creating the posterior probabilities for whether which keyword is said or it is another speech or noise. With time acoustic models had to meet the demands of increasing accuracy and decreasing computational cost. The latter requirement is due to the fact that most keyword algorithms run on battery-powered devices with limited computational power. If generating the posteriors requires a large amount of computation, it can be infeasible for mobile applications due to its large delay and

overconsumption of battery capacity.

The first DNN-based acoustic model was introduced in 2014. This fully connected feed-forward (FCFF) neural network has three hidden layers, each containing 128 neurons. This method performed 45% better than a competitive hidden Markov model and 39 % better in the presence of babble noise while having lower computational needs than the HMM baseline. [7].

Nowadays CNNs and RNNs have taken over fully connected DNNs in keyword-spotting applications due to their increased performance and decreased model size, hence computational demand.

One of the main attractive traits of convolutional neural networks is that they can exploit local time-frequency correlations. This makes them capable of outperforming simple FCFF neural networks. By changing the hyperparameters, such as kernel size or filter strides, CNNs are easily adjustable for various applications, thus increasing their accuracy without necessarily sacrificing their performance. By employing residual connections, the number of parameters can be substantially decreased, thus further lowering the computation without compromising accuracy [15]. Convolutional neural networks apply one or two-dimensional convolutions to learn acoustic features.

One-dimensional CNNs apply convolution on the time axis, also called temporal convolution [15]. 1D CNN keyword spotting systems require much less computation than convolutions in two dimensions, making them greatly more usable on mobile devices. Choi et al. [10] used a one-dimensional convolutional neural network for keyword spotting speeding up the keyword spotting process more than 385 times. Another reason why such models are so efficient is due to the fact that they do not require the input data to be converted into spectrogram images, thus further reducing the computations.

Based on image analysis, 2D CNNs are fed with spectrogram images of the input data exploiting the previously mentioned local time-frequency correlations. Two-dimensional neural networks can be fed with mel-scale spectrograms closely modeling human hearing.

Recurrent neural networks have also been used several times for KWS outperforming FCFF neural networks. LSTM neural networks overcame the problem of vanishing and exploding gradients [26]. Recurrent neural networks can exploit short and long-term dependencies. CNNs and RNNs can be combined into recurrent convolutional neural networks (RCNNs) [15]. Zeng and Xiao [59] used RCNN for KWS tasks achieving state-of-the-art performance at the time with a lightweight neural network model.

Another new method for acoustic models uses Transformer-based neural network models. A keyword transformer (KWT) learns features from two-dimensional mel-spectrograms while learning temporal, temporal dependencies through self-attention [4]. Keyword transformers are discussed in detail in chapter 3.

Acoustic models for keyword spotting are still a subject of numerous research. More and more models and training methods are presented every year that increase accuracy, decrease model size and computation, or both.

2.2.3 Posterior Handling

After the acoustic model produces the output probabilities $\mathbf{y}^{\{i\}}$, they must be processed according to certain criteria. Two major methods of posterior handing are distinguished: streaming mode (static) and non-streaming mode (dynamic) [15].

Non-Streaming Mode

Non-streaming mode or static posterior handling is simply a multi-class classification problem. The keywords are loaded in separate audio files, usually 1s long, and their labels are predicted as the highest value of the posterior probabilities. Static posterior handling is simpler than dynamic because it does not have to deal with inter-class transitions [15].

One, on the other hand, can argue about the validity of the non-streaming test of KWS systems since the practical application of KWS is, in fact, dynamic. The reason why there is so much work testing keyword spotting systems in non-streaming mode is due to its simplicity. On the other hand, López-Espejo et al. [15] found that there is a high correlation between the test results of streaming mode and non-streaming mode [38], [37].

Streaming Mode

Streaming mode or dynamic mode of posterior handling is the practical application of keyword spotting. The audio is continuously monitored in dynamic mode, wherein the keyword is not necessarily present. The KWS system has to handle whether a keyword is present in the audio or some other speech is spoken, or there is only noise or silence.

In streaming mode, the system continuously puts out a sequence of posterior probabilities $\{... \mathbf{y}^{\{i-1\}}, \mathbf{y}^{\{i\}}, \mathbf{y}^{\{i+1\}} ...\}$ in every given time segment. These posteriors are usually very noisy, and a smoothing window is used to reduce the peaks [50]. Assuming that every posterior $\mathbf{y}^{\{i\}}$ represents the probabilities of N keyword, by smoothing those posteriors, it can be used to detect whether a keyword is present at all in the audio stream. That can be done by comparing the value of the probabilities with a sensitivity threshold [15]. If the probabilities are even, they are under that threshold, but if the probability of a certain word peaks out, then it usually exceeds the threshold, and the keyword is spotted.

Another problem that can arise in streaming mode is that one keyword can activate multiple times. That can be alleviated by simply waiting for a short time after a KWS system is triggered [15].

Although streaming mode posterior handling directly represents the practical applications, most of the studies prefer non-streaming mode benchmark evaluation of models due to the latter's simplicity.

2.3 Robustness of Keyword Spotting

Several studies of KWS evaluate their KWS model on commonly used benchmarks such as Librispeech [42], or Google Speech Commands Dataset (GSCD) v1 [56] and v2 [55]. Such datasets make unequivocal non-streaming comparisons between models. Google Speech Commands Dataset is discussed in detail in section 6.1.

Keyword spotting algorithms are trained on the training set and evaluated on the test set of the dataset. In theory, the training set and the test set represent the same domain of data. In practice, however, this domain can shift for various reasons. Domain shift can be caused, for example, by gender (female or male voice), age (a child's voice sounds different from a mature person's), accent, or other physical conditions such as reverberation or background noise. A good keyword spotting system has to function the same in the above-mentioned conditions, i.e., it has to be robust for those conditions. The rest of the section discusses the robustness against background noise.

There are several methods to enhance the robustness of such KWS algorithms. López-Espejo et al. distinguish two main groups of handling noise robustness: front-end and back-end methods [15].

2.3.1 Front-End Methods

Front-end methods are applied to the input to reduce the noise before feeding it to the acoustic model, thus increasing the robustness of the system. There are several methods to reduce noisy features and enhance speech features. This section discusses two methods: DNN Feature Enhancement and Adaptive Noise Cancellation.

DNN Feature Enhancement

Deep Neural Networks can be trained to separate speech features from noise ones, thus cleaning the input data. The output of such a network becomes the input of the KWS acoustic model.

Gu et al. [21], for example, use a DNN-based speech enhancement (SE) algorithm that is fed with the log-spectrogram of the input noisy audio, which tries to predict a denoising mask. The mask is multiplied with the input (noisy) spectrogram, and the product is fed to a convolutional neural network which makes the acoustic model of the KWS system [21].

In another approach, Jung et al. use two cascaded dilated CNNs with residual connections to estimate the spectral distortion of the noisy input and then subtract the estimate from the input [31].

Adaptive Noise Cancellation

Adaptive noise cancellation (ANC) is widely used in headphones and earphones for undisturbed listening experiences or phone calls, and it can also be used for keyword spotting with great success. The ANC algorithms, which are usually recursive least squares (RLS) [28], [29] or Wiener filter-based [29], [40], are run before the acoustic model, and their weights are updated based on either the input [29] or the results of the acoustic model [28].

2.3.2 Back-End Methods

Back-end methods are applied to the acoustic model to increase its prediction accuracy in various noisy scenarios. This section discusses multi-style training and adversarial training.

Multi-Style Training

Multi-style training is one of the simplest ways to increase the robustness of the network against background noise and reverberation. Noise contamination can be done by adding background noise to the training data, which is expected to be found later in the test data or practical application. Noise is added with different signal-to-noise ratios (SNR), usually in the range of -10 dB and 20 dB [37]. Adding noise to the training data is also referred to as data augmentation. Another popular distortion method is done by SpecAugment, which applies certain time and frequency masks on the input data on the log-mel spectrogram of the input audio [43]. Data augmentation is used to increase robustness not only in noisy environments but ideal environments too.

Adversarial Training

Many neural networks are prone to misclassify keywords (or images in other applications) when small but deliberate perturbations are added to them. These are called adversarial examples [19]. To alleviate the misclassification rate, neural networks can be trained or retrained with adversarial examples, thus smoothing the predictions and hence, the false alarms [54].

In another work, Larsen et al. use adversarial multi-task learning to improve the robustness of a CNN-based voice activity detection (VAD) model [34]. The model consists of an encoder block, a framing block, and a decoder block. During training, a discriminative network is added after the framing block, parallel to the decoder block. The task of the discriminative network is to predict the type of noise that was added to the input audio. During backpropagation, the gradients are multiplied by a negative factor α in the adversarial path. The idea behind this method is that framing block produces noise-invariant features [34]. This method is explained in section 4.3.

Chapter 3

Baseline Models

Many studies have been conducted about the robustness of various CNN and RNN-based keyword spotting systems. In 2017 Vaswani et al. [53] published the Transformer network, which was originally developed for text translation, but soon was adopted to different domains, such as image [44], speech [2] and, in particular, keyword spotting [4].

Transformer networks can achieve state-of-the-art or competitive performance without pretraining. However, self-supervised pretraining has been shown to increase their performance even further. These pretraining methods were, nonetheless, modality-specific. By pursuing modality-agnostic self-supervised pretraining, data2vec [1] was released in 2022. Data2vec trains a Transformer network in a teacher-student mode. In another study, Bovbjerg and Tan [6] showed that pretraining a Keyword Transformer in a data2vec model could substantially increase the performance of a KWT in label-deficient applications.

In this section, the Keyword Transformer and data2vec will be introduced, followed by the above-mentioned study about pretraining Keyword Transformers with data2vec.

3.1 Transformer

The Transformer network was introduced by Vaswani et al. in 2017 [53]. The model was originally developed for translation tasks, particularly English-to-German and English-to-French in their work, and improved over the best already existing model in both tasks. Self-attention had been previously used in other models, mostly in recurrent neural networks. The Transformer network was the first that relied entirely on a self-attention mechanism, omitting recurrence. That let the model exploit parallelism, thus significantly improving performance. Self-attention is a positional attention mechanism between tokens in the input.

3.1.1 Model Structure

The original Transformer, which was designed for text translation, had an encoder-decoder structure. The encoder takes an input sequence $(x_1, \ldots, x_n)$ and generates a sequence of continuous representations $(z_1, \ldots, z_n)$. By receiving the encoder's output, the decoder generates an output sequence $(y_1, \ldots, y_n)$.

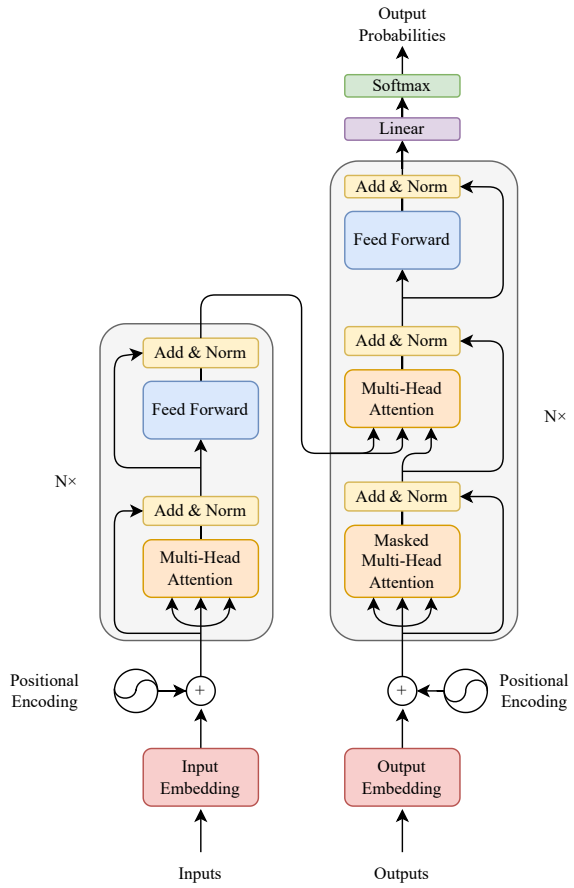

Figure 3.1: General encoder-decoder structure of the Transformer network [53]

The encoder builds from six identical blocks. The input is fed into an embedding block, so the data can be processed by the neural network. A positional encoding is added to the input data, which is needed for self-attention. The positional encoded data then goes into a multi-head attention block. Self-attention is an attention mechanism that exploits the positional relationship between the input tokens. The multi-head attention block is followed by a fully connected feed-forward network block. Both sub-blocks are followed with a layers normalization, and residual connections are applied around the sub-blocks [53].

The decoder block has a similar structure as the encoder, also containing six identical blocks. A masked multi-head attention block is inserted before the first multi-head attention block, followed by a layer normalization. Residual connection is also applied around the masked multi-head attention block. The masking is used in a way that the for each position, the encoder is prevented from attending to subsequent positions. By this, the prediction for a position i depends only on the previous values at the position less than i. The last decoder block is followed by a linear block with a softmax activation function to predict the next token of the output sequence [53].

3.1.2 Multi-Head Attention

As mentioned before, the heart of the Transformer network is the self-attention block or, as the authors of [53] call it: scaled dot-product attention. The scaled dot-product attention consists of key and query vectors in the dimension of d_k and d_v. The dot-product of the vectors is computed and then scaled back by $\sqrt{d_k}$ to keep the softmax function away from its boundaries. Then an optional mask is applied to the product by setting the masked-out elements to $-\inf$. After the additional masking, a softmax function is applied. By the nature of softmax (Eq. 3.1), elements with a value of $-\inf$ will have an output value of zero [53].

$$\sigma(\mathbf{z}) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{3.1}$$

Multiplying the result of the softmax function with the values, the weights of the values are computed. In practice, the queries, keys, and values are packed together into Q, K, and V matrices to exploit faster computation. Thus the attention head becomes

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.2}$$

Multi-head attention is done by linearly projecting the queries, keys, and values by learned projection matric W, exploiting parallel computation. The dimensions of the projection matrices are d_k, d_k, and d_v for queries, keys, and values, respectively. Thus, multi-head attention becomes [53]:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{3.3}$$

where

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{3.4}$$

The dimensions of the projection matrices are $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, and $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$.

Another key element of Transformer networks that is used with self-attention is positional encoding. Since the network does not contain recurrence or convolution, positional encoding is needed for extracting information from the order of the sequence. Vaswani et al. used sine and cosine functions with various frequencies as positional encoding functions, which according to their experiments, produced similar results as a learned encoding (Eq 3.5) [53].

$$\begin{aligned} PE_{(\text{pos},i2)} &= sin(pos/1000^{2i}/d_{\text{model}}) \\ PE_{(\text{pos},i2+1)} &= cos(pos/1000^{2i}/d_{\text{model}}) \end{aligned} \tag{3.5}$$

Self-attention is beneficial in making the model learn long-range dependencies, which has been a challenge in several machine learning models. Moreover, self-attention can be calculated in parallel in contrast to sequential computations of recurrent neural networks. Self-attention layers require less computation than recurrent or convolutional layers when the sequence length is less than the model dimension [53].

3.2 Keyword Transformer

Thanks to its novelty, the Transformer network was soon adopted in other domains, such as image [44] or speech [2]. Inspired by the performance of the Vision Transformer, Berg et al. developed the

Keyword Transformer (KWT) [4], a transformer model specialized for keyword spotting. In their study, they only tested the model in non-streaming mode but acknowledged that the Transformer network had been previously used in streaming settings. Their model set new benchmark records for Google Speech Command Dataset [55] while achieving substantial latency and energy reduction when on a mobile phone device.

3.2.1 Model Structure

The Keyword Transformer model follows the model structure mentioned in section 2.2. It consists of a feature extractor block, an acoustic model, and a posterior handling block. Figure 3.2 shows the full model structure of a Keyword Transformer.

Feature Encoder

Before feeding the acoustic model, the audio is converted into MFCC spectrograms. The spectrograms are computed from 30 ms long windows with a stride of 10 ms. The number of mel features is set to 40. The authors of [53] also use data augmentation, which consists of shifts in the time axis, resampling, adding background noise as well as applying SpecAugment [43] on the MFCC spectrograms. After generating the Mel-Spectrograms, smaller patches are created by dividing the spectrogram into N pieces.

Acoustic model

The base of the acoustic model is an encoder-only Transformer architecture. To feed the encoder with the right data, the MFCC spectrogram $X \in R^{T \times F}$ (where $t = 1, ..., T$ refers to the time windows and $f = 1, ..., F$ refers to the frequency domain) is first mapped to a higher dimension d with a projection matrix $W_O \in \mathbb{R}^{F \times d}$. After that, a learnable class embedding $X_{\text{class}} \in \mathbb{R}^{1 \times d}$ is attached in the time domain. As mentioned before, in order for the Transformer to learn dependence in the time domain, the data should be contextualized by adding positional embedding $X_{\text{pos}} \in \mathbb{R}^{(T+1) \times d}$ for each spectrogram patch [4].

The Transformer block contains L multi-head self-attention (MSA) and multi-layer perception (MLP) blocks. The queries, keys, and values for the l^{th} attention head are calculated as $Q = X_l W_Q$, $K = X_l W_K$, and $V = X_l W_V$ where W_Q, W_K, and W_V are $d \times d_h$ matrices, where d_h is the dimension of the self-attention head. The attention and the multi-head attention are calculated the same way as in [53], which are

$$\text{SA}(X_l) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_h}}\right) V \tag{3.6}$$

and

$$\text{MSA}(X_l) = [\text{SA}_1(X_l); \text{SA}_2(X_l); ...; \text{SA}_k(X_l)] W_P \tag{3.7}$$

where $W_P \in \mathbb{R}^{kd_h \times d}$ is another projection matrix.

After the MSA block layer norm (LN) is applied, followed by an MLP and another LN block. Same as for the original Transformer, residual connections are applied around the MSA and the MLP blocks.

Unlike in Visions Transformer [12], where the attention occurred across the image patches, in KWT, the attention is only implemented in the time domain. Berg et al. implemented three different transformer models based on the number of attention heads, which was set to $k = 1, 2, 3$.

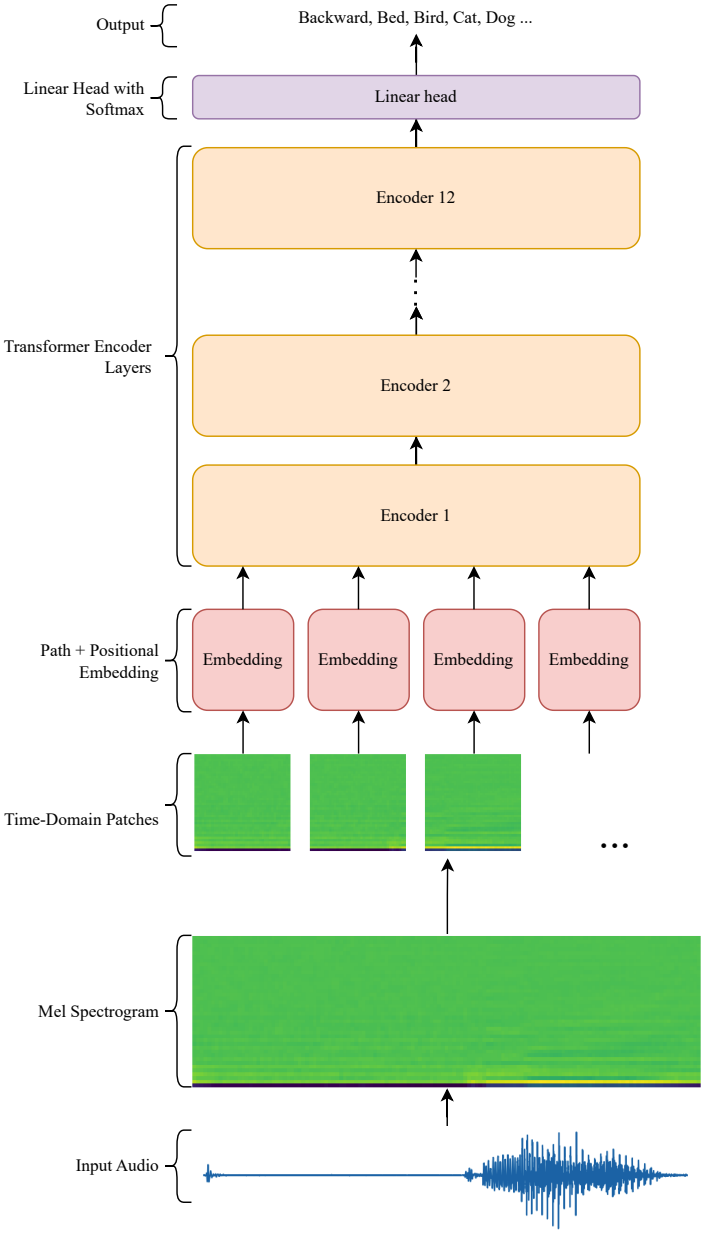

Figure 3.2: Keyword Transformer structure [4]

Model	dim	mlp-dim	head	layers	# parameters
KWT-1	64	256	1	12	607K
KWT-2	128	512	2	12	2394K
KWT-3	192	768	3	12	5361K

Table 3.1: Model parameters of the KWT architectures [4]

Model	V1-12	V2-12	V2-35
MHAtt-RNN	**97.50 ±0.29**	98.36 ±0.13	97.27 ±0.02
KWT-3	97.24 ±0.24	**98.54 ±0.17**	97.51 ±0.14
KWT-2	97.36 ±0.20	98.21 ±0.06	**97.53 ±0.07**
KWT-1	97.05 ±0.23	97.72 ±0.01	96.85 ±0.07

Table 3.2: Test results of Google Speech Command Dataset V1 [56], and V2 [55] in 12 and 35-word mode based on the training and evaluation by Berg et al. [4]

They fixed the number of transformer blocks to 12 and the dimension-to-head ratio to $d/k = 64$. Table 3.1 concludes implemented transformer models.

Posterior Handling

The model was evaluated in non-streaming mode on Google Speech Commands Dataset V1 [56] and V2 [55]. The model was trained and tested as a simple keyword classifier of 30 and 35 different words for V1 and V2, respectively.

3.2.2 Performance

The Keyword Transformer had a competitive performance against state-of-the-art models, such as MHAtt-RNN [47], when trained and evaluated on the Google Speech Commands Dataset V1 and V2 [4]. Since transformers tend to benefit from larger datasets, the KWT models could not surpass the accuracy of MHAtt-RNN on smaller datasets, such as 12-word GSCD V1. On larger datasets, such as GSCD V2 12 and 35-word configuration, KWT3 achieved better results than MHAtt-RNN, and KWT-2 had higher accuracy than MHAtt-RNN on GSCD V2 35-word configuration. Table 3.2 shows the results of the accuracy of the Keyword Transformers developed by Berg et al. [4].

3.3 Data2vec

While supervised training requires human-labeled datasets, self-supervised training does not, which leads to access to much larger and more affordable datasets. Transformer network generally benefits from larger datasets [4], and several studies have shown that self-supervised pretraining can lead to better performance on the downstream task. Self-supervised learning has shown great results in computer vision [12], in natural language processing [11], or in speech processing [2].

Data2vec is a modality-independent self-supervised learning model trained either in a teacher or student mode. It has shown excellent results in computer vision, NLP, and speech, by achieving state-of-the-art or competitive performance over modality-specific models.

3.3.1 Model Structure

Data2vec trains an off-the-shelf Transformer network either in student or teacher mode. In teacher mode, the network builds representations of the full input data. In student mode, the network receives a masked version of the input data and tries to predict the representation of the full data by predicting the average of the top K layers of the teacher. The weights of the teacher are an exponentially moving average (EMA) of the student weights. In contrast to other models, such as wav2vec [2] or HUBERT [27], where discrete units are learned, data2vec learns continuous latent representations of the input data, which is contextualized through self-attention [1].

Although the training targets are created the same way in all modalities, the encoding of the data is modality specific. For computer vision, following the ViT strategy, linearly transformed 16×16-pixel patches of the input image are fed into the network. For speech data, a 1D convolutional network converts the 16 kHz waveform to 50 Hz representations into sub-word units and maps them to vectors via learned embeddings [1].

In student mode, the masking is done after the embedding. For computer vision, block-wise masking is done where arbitrary aspect ratios of adjacent patches are masked. For speech, embedded speech representations are masked out, and for text, the masking is done on the embedded tokens [1].

The targets are formed from the average of the outputs of the top K layers of the teacher network for timesteps (or image patches) where the input is masked out. Due to the embedding, the targets are contextualized and continuous. Before averaging, normalization is applied as

$$y_t = \sum_{l=L-K+1}^{L} \hat{a}_t^l \tag{3.8}$$

for L blocks and where $\hat{a}_t^l$ is the block l at time-step t. Normalization is done to prevent the model from collapsing. Averaging of the outputs performs the same and costs less computation than predicting the outputs for each layer [1].

The prediction of continuous values is a regression task, and for the objective, a Smooth L1 loss function is used:

$$\mathcal{L}(y_t, f_t(x)) = \begin{cases} \frac{1}{2}(y_t - f_t(x))^2/\beta & |y_t - f_t(x)| \leq \beta \\ (|y_t - f_t(x)| - \frac{1}{2}\beta) & \text{otherwise} \end{cases} \tag{3.9}$$

where β is a tunable parameter that sets the transition between the squared loss and the L_1 loss depending on the difference between the predictions and the targets [1].

The teacher tracks the student parameters by an exponential moving average of the student parameters. Thus the weights of the teacher network are updated as

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

where τ is scheduled from τ_e to τ_n and kept constant afterward [1].

3.4 Pretraining Keyword Transformer with Data2vec

Keyword Transformer showed great potential in keyword spotting. It was also found by Berg et al. that transformer network benefits from large datasets [4]. Their results showed that Keyword Transformer could not outperform MHAtt-RNN on the smaller Google Speech Command Dataset

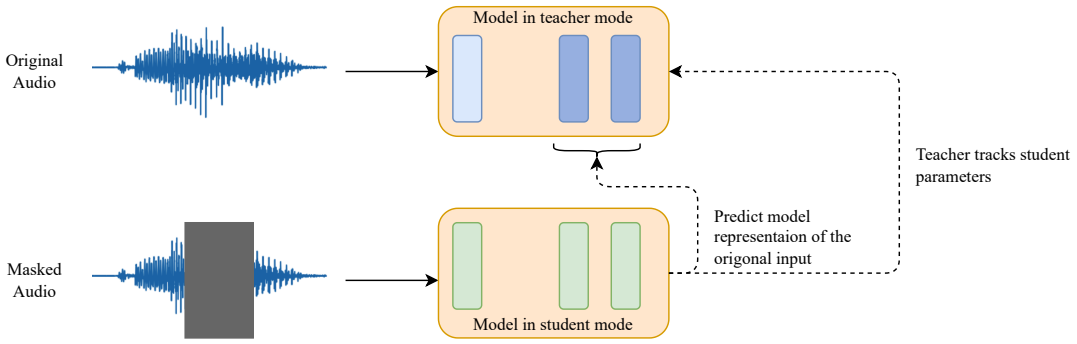

Figure 3.3: Data2vec model structure [1]

V1. However, when the models were trained on larger datasets, then the larger Keyword Transformer models could outperform MHAtt-RNN [4]. Berg et al. used knowledge distillation [22] to improve the accuracy of the KWT models. This method still requires another model trained on a possibly large dataset.

Many practical applications, however, can not get access to large human-labeled datasetsself-supervised pretraining aims to solve this problem. Data2vec has shown that self-supervised pretraining can improve the downstream tasks of the neural networks in several modalities. Bovbjerg et al. [6] apply data2vec pretraining on a Keyword Transformer in a label-deficient scenario. They train and evaluate their model on a label-deficient Google Speech Commands Datasets V2 in 35 keyword mode by randomly selecting 80% of the training data as a label-deficient pretraining set [6]. Such as [4], the evaluation was done only in non-streaming mode. (They additionally pretrained the models with LibriSpeech [42]).

3.4.1 Model Structure

For feature extractions, MFCC spectrograms were used for feature extractions, with 40 MFCC features, a window length of 480 samples, and a hop length of 160. This is identical to 30 ms window length and 10 ms stride for a sampling rate of 16 kS/s. For the positional and class embedding, they followed the method described in the KWT study [4].

To form a baseline, the Keyword Transformers listed in Table 3.1 were trained on the label-deficient dataset in supervised mode. The models were trained for 140 epochs, with a batch size of 512. The parameters were optimized with Adam Optimizer [35], and the learning rate was warmed up in a linear increment for the first 10 epochs as

$$\eta_0 = \frac{\eta_{\max}}{\text{batch size} \cdot n_{\text{epochs}}} \tag{3.11}$$

Which is followed by a cosine annealing schedule [36]. Following the hyperparameter settings of the KWT study [4], weight decay of $\lambda = 0.1$ and label smoothing [41] of $\epsilon = 0.1$ was used.

The transformer networks were pretrained on the label-deficient training dataset in the teacher-student mode. The student received the masked input and predicted the latent representation of the whole input encoded by the teacher. The masking was done in the time domain, where the position of the mask was randomly selected in the time axis, and the 10 consecutive MFCC vectors were masked out (replaced with a *MASK* token embedding). Roughly 65% of the input spectrogram was masked out [6].

Model	Parameters	Baseline	Data2vec pretraining
KWT-1	600k	0.8572	0.9394
KWT-2	2.4M	0.8584	0.9507
KWT-3	5.4M	0.8411	0.9527

Table 3.3: Baseline and pretrained-finetuned KWT models [6]

Most of the hyperparameters were chosen according to the data2vec study [1]. However, some of the parameters were changed. The batch size was set to 512, and the number of epochs was set to 200. The EMA decay steps were set to 1000. MSE was used for the objective function, and the weights were optimized with Adam optimizer with a learning rate of 5×10^{-3}, weight decay of 10^{-2}, and 1-cycle learning rate schedule [48], [6].

3.4.2 Results

The baseline models were trained on 20% of the training data. Then the models were pretrained on the unlabeled pretraining dataset (80% of the original training set) and finetuned on the labeled training data. Their experiments found that pretraining substantially increased the accuracy of the predictions, as Table 3.3 shows.

Based on their experiments, KWT-3 performed slightly worse than KWT-2 and KWT-1; however, it performed the best when pretrained. Their experiments unequivocally showed that Keyword Transformers benefit from self-supervised pretraining.

Chapter 4

Improving Robustness of Speech Recognition Systems

As mentioned in section 2.3, there are several methods to improve the robustness of keyword spotting. On the other hand, some of the methods were only applied to other ASR tasks. Since keyword spotting is a subproblem of automatic speech recognition, it is worth assuming that practices that improve over other ASR subproblems (e.g., voice activity detection [34]) can also enhance keyword spotting.

This chapter focuses only on backend methods of improving KWS and other ASR algorithms. In this chapter, other studies are introduced briefly about self-supervised and supervised methods. Some methods are adopted to enhance Keyword Transformer, and others are mentioned for being relevant.

4.1 Enhanced Wav2vec

Zhu et al. [60] observe that wav2vec2.0 [2] performs better on noisy test data when it is pretrained on also on a noise set. This, however, came with the price of decreased performance in clean applications. To strengthen the robustness and anneal the beforementioned negative effect, Zhu et al. proposed the enhanced wav2vec2.0 model, which showed better performance than the original wav2vec2.0 while only having a minimal decrease in performance in clean tests [60].

Their proposed model is depicted in figure 4.1 consists of a shared feature encoder that maps the input audio into latent space via a 7-layer CNN network such that $f : X \rightarrow Z$, where X is the input and Z is the encoded data. The model is jointly trained on clean and noisy audio. The Transformer receives the encoded noisy waveform and produces outputs of C such that $g : Z \rightarrow C$. The encoded clean waveform is fed into a clean vector quantization module to form the discrete targets for the Transformer, such that $VQ : Z \rightarrow q$. The goal is that the Transformer learns clean speech representations from noisy speech [60].

The output of the Transformer becomes

$$C_{\text{noisy}} = g(f(X_{\text{noisy}})) \tag{4.1}$$

and the quantized targets become

$$q_{\text{clean}} = VQ(f(X_{\text{clean}})) \tag{4.2}$$

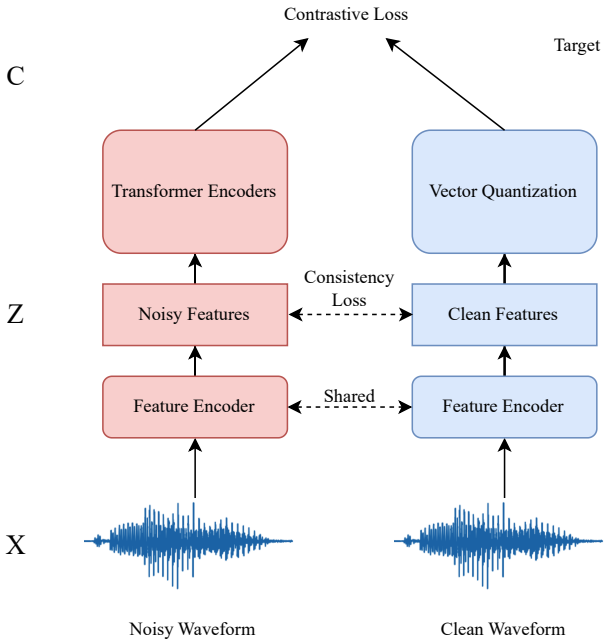

Figure 4.1: Structure of the enhanced wav2vec2.0 model [60]

The VQ module maps the clean features to $\mathbf{l} \in \mathbb{R}^{G \times V}$, where G is the number of codebooks and V is the number of entries. A Gumbel-softmax function [30] is applied to select discrete codebooks. For every time t, one entry e_t is selected, then the entries are concatenated, and a linear transformation is applied to obtain q_{clean}.

The overall loss function L consists of the contrastive loss L_m, the diversity loss L_d (which increases quantized codebook features), and l_2 penalty function L_f, and the consistency loss L_c between the clean features and the noisy features, which is also an l_2 norm function. The loss function L thus becomes [60]:

$$L = L_m + \alpha L_d + \beta L_f + \gamma L_c \tag{4.3}$$

where

$$L_m = -\log \frac{\exp(\text{sim}(C_{\text{noisy}_t}, q_{\text{clean}_t})/\kappa)}{\sum \hat{q}\ Q_t \exp(\text{sim}(C_{\text{noisy}_t}, \tilde{q})/\kappa)} \tag{4.4}$$

$$L_d = \frac{1}{GV} \sum_{g=1}^{G} \sum_{v=1}^{V} \bar{p}_{g,v} \log \bar{p}_{g,v} \tag{4.5}$$

$$L_c = ||Z_{\text{noisy}} - Z_{\text{clean}}||_2 \tag{4.6}$$

and α, β, γ are parameters. In contrastive loss, $\tilde{q}$ is a quantized candidate feature, κ is a temperature parameter, and sim() refers to the cosine similarity function. In the diversity loss, $\bar{p}_{g,v}$ is the probability of choosing the v-th codebook in the group g, and is calculated as

$$\bar{p}_{g,v} = \frac{\exp(\bar{l}_{g,v} + n_v)/\tau}{\sum_{k=1}^{V} \exp(\bar{l}_{g,k} + n_k)\tau} \tag{4.7}$$

where τ is a temperature parameter, $\bar{l}_{g,v}$ is the average logits across utterances in the batch [60].

In the pertaining, the feature encoder consists of 7 layers of convolutional modules with a channel number of 512. The window covers 25 ms of audio with a hop length of 20 ms. the Transformer encoder consists of 12 layers of Transformer blocks with 8 heads, and the dimension of the MSA is 512. For the VQ module, the number of codebooks is set to $G = 2$, the number of entries is $V = 320$, and the dimension of each entry is set to 128. The model contains approximately 45M parameters. For the loss functions, the temperature parameters κ and τ are annealed from 2 to 0.5. The learning rate is warmed up to 5×10^{-4} in the first 8% of the epochs and then decayed linearly [60].

During fine-tuning, the output linear head is removed, and the model is finetuned with labeled data with 30 characters, wherein 26 are letters, and 4 are special characters. The objective function is a connectionist temporal classification loss function [20] [60].

In their experiments, their model performed best in comparison to other models, such as DEMUCS [14], Avt [46], and the original wave2vec2.0, both for stationary and non-stationary noisy tests when pretrained and finetuned with FreeSound-augmented speech data. The model performed worse on non-stationary noise than on stationary. For clean speech, wave2vec 2.0 performed worse when trained on noisy data compared to the clean trains; more precisely, the word error rate (WER) was 11.0 when trained on clean speech and 25.0 when trained on noisy speech. It had a WER of 14.0 when pretrained on clean speech and finetuned on noisy speech and 13.5 WER when pretrained and finetuned on noisy speech. In contrast, enhanced wav2vec2.0 had 12.3 WER when pretrained and finetuned on noisy speech, which is better than the original wav2vec2.0, while performing substantially better on noisy tests [60].

Their experiments showed that enhanced wav2vec2.0 with modified pretraining can increase the performance of the network in noisy conditions while minimally decreasing the performance in clean speech. Their study also showed that pretraining with contrastive, consistency, and diversity loss functions can lead to better performance than simple pretraining on noisy data.

4.2 Robust Data2vec

In that study, Zhu et al. proposed a noise-robust data2vec pretraining framework by jointly optimizing a regression task and contrastive learning [61] for an automatic speech recognition task. Figure 4.2 depicts their proposed model, which consists of a student and a teacher transformer network. The student network receives noisy x_{noisy}, and the teacher receives a clean x_{origin} waveform. The inputs are mapped to the latent space z_{noisy} and z_{origin} of a feature encoder before the transformer blocks. Following the method of data2vec study [1], the student tries to predict the average of the output of the top M transformer layers of the teacher $c^{l}_{\text{origin}_t}$. The weights of the teacher are updated as the EMA of the student, as in Eq. 3.10. The target as the average output of the top M layer is calculated as

$$c_{\text{tar}_t} = \frac{1}{M} \sum_{l=L_M+1}^{L} c^{l}_{\text{origin}_t} \tag{4.8}$$

The model is optimized over the regression and the contrastive losses. The regression loss is described as

$$\mathcal{L}_{\text{reg}} = \begin{cases} \frac{1}{2}(c_{\text{pre}_t} - c_{\text{tar}_t})^2 / \beta & |c_{\text{pre}_t} - c_{\text{tar}_t}| \leq \beta \\ \left(|c_{\text{pre}_t} - c_{\text{tar}_t}| - \frac{1}{2}\beta\right) & \text{otherwise} \end{cases} \tag{4.9}$$

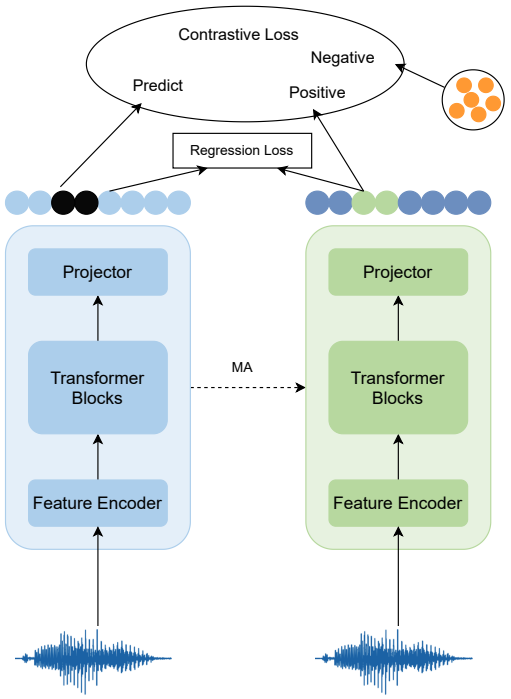

Figure 4.2: The structure of robust data2vec [61]

where β is a controlling parameter of the transition from squared loss to L1 loss. The contrastive loss is

$$\mathcal{L}_c = -\log \frac{\exp(\text{sim}(c_{\text{pre}_t}, c_{\text{tar}_t})/\kappa}{\sum_{\tilde{c} \in \{c_{\text{tar}}, c_n, c_{ns}\}} \exp(\text{sim}(c_{\text{pre}_t}, \tilde{c})/\kappa)} \tag{4.10}$$

The resulting loss becomes

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{reg}} + \lambda \mathcal{L}_c \tag{4.11}$$

where λ is a hyperparameter.

The authors aimed the model to learn high-level non-semantic noise robust representations of the speech. To achieve this, they created noisy perturbed samples by slicing the output into patches along the time and dimension axes. The non-semantic contrastive loss thus becomes

$$\mathcal{L}_c = -log \frac{\exp(\text{sim}(c_t, c_{p_t})/\kappa)}{\sum_{\tilde{c} \in \{c_p, c_n, c_{ns}\}} \exp(\text{sim}(c_t, \tilde{c})/\kappa)} \tag{4.12}$$

where c_t, c_p, c_n, and c_{ns} are the representations of the query samples, positive samples, negative samples, and non-semantic negative samples, κ is a temperature coefficient, and sim() is the cosine similarity function. ’

By analyzing the cosine similarities of the predicted positive and the predicted negative samples, the authors removed the more distinguishable negative samples by removing k samples with the smallest cosine similarity score out of N samples.

The model was trained on the CHiME-4 [39]. The noise dataset was used from the MUSAN dataset [49], and it was applied during the pretraining. In order to create noisy speech, the speech data was mixed with noisy data at an SNR level from 0 to 25 dB [61].

Model	LM	WER	
		dt05_real	et05_real
Wav2vec2.0 Base [2]	None	10.6	17.6
	LSTM	3.7	7.2
HuBERT Base [27]	None	10.4	17.0
	LSTM	3.8	7.1
Enhanced Wav2vec2.0 [60]	None	9.4	15,7
	LSTM	3.5	6.4
Data2vec Base [1]	None	9.5	15.7
	Transformer	3.5	6.5
Robust Data2vec [61]	None	8.3	12.8
	Transformer	**3.1**	**5.8**

Table 4.1: WER comparison between robust data2vec and other models in CHiME-4 dataset

The model consists of 12 layers of transformer network with a dimension of 768 and a final feed-forward network with a dimension of 3072. The feature encoder has 7 layers of convolutional network with kernels of (10, 3, 3, 3, 3, 2, 2) and strides of (5, 2, 2, 2, 2, 2, 2). In joint training, $\tau_0 = 0.999$, $\tau_e = 0.9999$ and the number of increments to $\tau_n = 30000$ (Eq. 3.10. The masking probability is set to $p = 0.065$, and the following 10-time steps are masked out. 50 negative samples are adopted, with additional 50 non-semantic negative samples. During training, the student learns the average of the outputs of the top 8 Transformer layers of the teacher. In Eq. 4.11 λ is set to 1.0 [61].

After pertaining, the Transformer model is finetuned on the labeled set of CHiME-4 5k vocabulary. It was tested on the development and evaluation subdirectories, dt05_real and et05_real, respectively. The authors compared their results with other models, including HuBERT [27], wav2vec2.0 [2], enhanced wav2vec2.0 [60], data2vec base [1], both with language models and without. They also compared their results with other supervised and self-supervised methods. Table 4.1 shows their results for the beforementioned modes.

Their experiments showed that data2vec performed better than wav2vec2.0 and HuBERT. Enhanced data2vec performed marginally better than data2vec, and robust data2vec had the lowest WER both for dt05_real and et05_real. Enhanced Wav2vec2.0 and robust data2vec both show that contrastive self-supervised learning has great potential to enhance the robustness of ASR tasks.

4.3 Adversarial Training for Voice Activity Detection

Voice Activity Detection (VAD) is a binary classification problem that predicts whether a human voice is present in the audio stream or not. It is usually used in front of other ASR tasks, such as speech enhancement [25] or speaker verification [9].

In the VAD study, Larsen et al. [34] studied adversarial training to enhance the robustness of a proposed voice activity detection system [58], which performed state-of-the-art results on AURORA2 [24] dataset.

The pipeline of the adversarial VAD is shown in Figure 4.3. The model is a fully CNN-based network, which performs 1D, i.e., temporal convolution, in contrast to 2D or time-frequency-based convolution methods to reduce the computational cost. The model consists of an encoder block

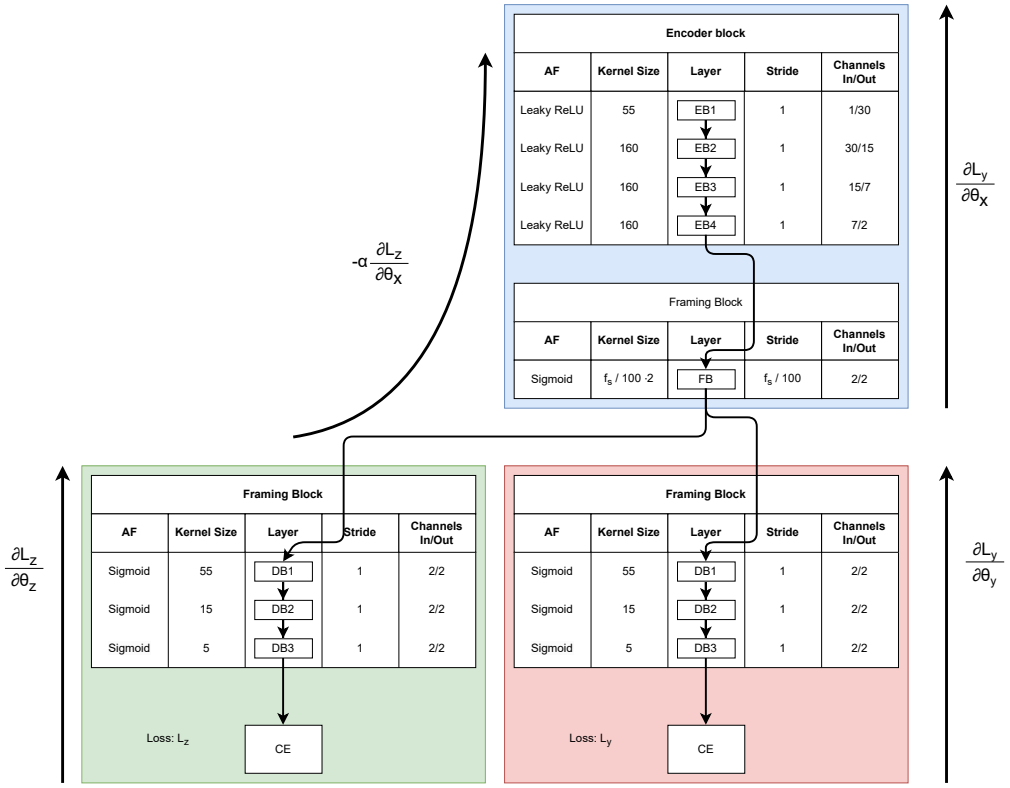

Figure 4.3: Adversarial VAD model [34]

(EB), a framing block (FB), and a decoder block (DB). The encoder block has 4 CNN layers with kernel sizes of (55, 160, 160, 160) and strides of 1 in every layer. The framing block with a kernel size of $f_s/100 \cdot 2$ and stride of $f_s/100$ generates features every 10 ms. The decoder block has 3 CNN layers with kernel sizes of (55, 15, 5) and strides of 1 in every layer. It purifies the feature and outputs a probability of whether the voice is present in the audio or not. The EB and FB have Leaky ReLU [57] activation functions, while the DB has sigmoid activation functions in its layers.

The authors, inspired by [17] added a new discriminative network (DN) parallel to the DB, whose hyperparameters are identical to the DB, except for the last layer, so the network can produce probability distribution over more (than two) discrete values. The task of the discriminative network is to predict the background noise, which was artificially added to the background audio. The output has $N+1$ channels, where N is the number of noise types the input is mixed with [34].

Two loss functions are calculated during training. The adversarial loss is calculated from the predicted and the target noise labels and cross-entropy loss functions were used, such as

$$\mathcal{L}_z = -\sum_i t_i \log(p_i) \tag{4.13}$$

The framing block does a binary classification over the existence of voice in the audio stream, for loss binary-cross-entropy was used, such as

$$\mathcal{L}_y = -[t \log(p) + (1-t)\log(1-p)] \tag{4.14}$$

where t is the true label, and p is the prediction.

The DN is optimized over $\mathcal{L}_z$, and the DB is optimized over $\mathcal{L}_y$. The EB and FB are optimized on both loss functions, but the gradients calculated from $\mathcal{L}_z$ are flipped over (multiplied by -1), so the encoder blocks and framing block are trained adversarially. The idea behind that is the model learns noise-invariant speech features in that way. The rate of adversarial gradient updated is controlled with the parameter $alpha$ by multiplying the partial derivative with it, such that:

$$- \alpha \frac{\partial \mathcal{L}_z}{\partial \theta x} \tag{4.15}$$

The value of $\alpha = 0.1$ was chosen empirically from the validation accuracy.

The model was trained and tested on the AURORA2 and TIMIT [18] datasets. To add background noise to the test set, *babble*, *bus*, *cafe*, and *pedestrian* noises were used from CHiME-3 [3] and *babble* generated by the authors of [33]. The noises were added in SNRs of (-5, 0, 5, 10, 15, 20) dB. With the exception of Clean and 20 dB on the AURORA2 dataset, models with adversarial training outperformed the baseline model with a maximum of 3.85% on ARURORA2-B [34].

Chapter 5

Enhancing Robustness of Keyword Transformer

This section discusses two main back-end methods of improving the robustness of Keyword Transformers, namely multi-syle training and adversarial training. Although there are many front-end methods to enhance the features of speech before feeding a neural network with them, this study focuses on the supervised and self-supervised pretraining and finetuning methods that are believed to enhance the robustness of KWTs. In this section, first, the proposed baseline model is introduced, and then two main methods will be explained that can be applied either during self-supervised pretraining or supervised finetuning.

5.1 Keyword Transformer

The baseline models Keyword Transformer and pretraining models of data2vec are adopted from the study of Bovbjerg [6]. During the study, experiments are conducted on KWT-1, KWT-2, and KWT-3 with the main parameters shown in Table 3.1. The main hyperparameters are identical to the models from [6] both for pretraining and finetuning.

5.1.1 Feature Extractor

The feature extraction is based on MFCC spectrograms, with the first 40 MFCC features used. The audio with a sampling rate of 16 kS/s is converted into mel-scale spectrograms with 30ms window length and 10ms stride. The size of the audio spectrograms thus becomes $40 \times T$, where T denotes the number of time steps during the formation of the spectrograms.

5.1.2 Acoustic Model

The Keyword Transformer provides the acoustic model with the parameters of 3.1. The MFCC spectrograms are first divided into patches of size $f \times t = 1 \times 40$, where f is the patch size in the frequency domain, and t is the patch size in the time domain. The patches are then embedded with a projection matrix $\mathbf{W}_0 \in \mathbb{R}^{(f \cdot t) \times d}$, where d is the embedding dimension and is equal to $64 \cdot k$, where

k is the number of attention heads. A class token $\mathbf{X}_{\text{CLS}}$ is then concatenated before the embedded spectrograms, and a positional encoding $\mathbf{X}_{\text{pos}}$ is concatenated after. The embedding thus becomes

$$\mathbf{X}_0 = [\mathbf{X}_{\text{CLS}}; \mathbf{X}\mathbf{W}_0] + \mathbf{X}_{\text{pos}} \tag{5.1}$$

The multi-head self-attention is described in Eq. 3.7.

In pertaining, the input of the student model is masked with a learnable MASK token embedding with a probability of p_{mask}, and next N_{mask} timesteps are masked-out. This method was adopted from wave2vec [2] [6].

5.1.3 Posterior handling

The model is trained and tested only in non-streaming mode. However, a strong correlation of performances between streaming mode and non-streaming modes was found by [15].

5.2 Multi-Style Learning

Multi-style learning has been successfully applied to several speech processing tasks, such as [32] or [45]. In multi-style training, the training data is mixed with background noise.

In this project, the models are only studied in non-streaming mode, which means that the keyword spotting is handled as a multi-class classification problem with keywords loaded from individual files. The noise is then added to each keyword in various SNR levels using either one noise type or a mixture of noise types. The authors of [32] found that training a network on a particular noise type, speaker (gender), or SNR gives better results than a mixture of noises, speakers, or SNRs, respectively.

In Keyword Transformer, multi-style training can be applied in self-supervised pretraining and supervised finetuning. Since, with clean data, the task of self-supervised pretraining is to learn speech features, the expected benefit of using noisy data in pretraining is questionable because the model probably learns noise features as well. In studies of enhanced wav2vec2.0 and robust data2vec, noisy data is applied during self-supervised pertaining, but the pretraining methods are modified by including a contrastive loss in the optimization phase.

During finetuning, however, including noise can be beneficial, so the model learns to find discriminative features, which are, in theory, noise-independent keyword-specific speech features. It is also believed that introducing self-supervised pretraining on clean datasets can further reinforce the model to learn discriminative speech features during the noisy finetuning process.

In keyword spotting, the model can be studied in speaker-specific, noise-specific, SNR-specific, or their opposite: speaker-agnostic, noise-agnostic, or SNR-agnostic conditions. Speaker-specific training means the model is trained on keywords spoken by humans belonging to one particular gender. Some datasets divide male and female voices into separate classes, which allows for studying the difference between the characteristics and their impact on noise-robustness. Other datasets, on the other hand, do not separate genders, and the distribution between genders is not necessarily 1:1. In this project, the models are trained on Google Speech Commands Dataset V2, which does not distinguish genders, and for that reason studying the impact of genders on noise robustness will be omitted.

5.3 Adversarial Training

Adversarial training has been used to enhance the robustness of speech recognition algorithms such as [34]. In this study, the base idea of [34] is applied to a Keyword Transformer both during pretraining and finetuning. Since the adversarial task of [34] is supervised, the self-supervised pretraining becomes semi-supervised.

5.3.1 Model Structure

The model has a Y-shaped structure. In the VAD study, the encoder block and framing block were shared, while the decoder block and the discriminative network stem from the framing block parallelly. Following this concept, the model consists of two keyword transformer encoders that share the feature encoder and the first K Transformer blocks. The original or friendly network is optimized over the initial keyword-spotting task, while the adversarial network is optimized over the noise-classification task. The shared layers are optimized over both tasks.

During training, the friendly task receives either clean or noisy data, while the adversarial task receives noisy data. The noise is added so that the ratio of the noisy and clean keywords is $N_{\text{noise}} : 1$, and the distribution of the noise types is uniform.

5.3.2 Adversarial Pretraining

During pertaining, the friendly part of the model consists of the student-teacher model of data2vec and is trained self-supervised on the label-deficient dataset. The student and teacher receive the masked and full versions of the input, respectively. The student predicts the averaged output of the top L layers of the teacher. The teacher's weights are updated as the EMA of the student's weights. After the last transformer block, the student network ends with a regression head.

The adversarial network has the same structure as the student encoder, with the exception of the regression head switched to a classification head. The adversarial network shares the feature encoder and the first K layers with the student network. During training, the network tries to predict the type of noise added to the pretraining data. This is a supervised task, which makes the self-supervised pretraining semi-supervised. The adversarial KWT produces a probability distribution over $N_{\text{noise}} + 1$ classes, where N_{noise} stands for the number of noise types and the +1 is for the clean data.

The friendly data2vec model is optimized over the data2vec regression loss $\mathcal{L}_f$ (Eq. 3.9). The loss for the adversarial task $\mathcal{L}_a$ is calculated with the cross-entropy loss (Eq. 4.13) since it solves a multi-class classification problem. The shared layers are optimized over both $\mathcal{L}_f$ and $\mathcal{L}_a$, where the gradients calculated from $\mathcal{L}_a$ are flipped and multiplied with α, which is a controlling parameter.

5.3.3 Adversarial Finetuning

During the finetuning process, both the friendly and the adversarial networks solve multi-class classification problems.

The friendly part consists of a Keyword Transformer encoder, which classifies the data based on the keyword. During training, the friendly network produces a probability distribution over N_{keywords} classes. The adversarial network is identical to the friendly network with the exception of the last MLP head, which adopts the number of noise types N_{noise} the network classifies. The friendly and adversarial networks share the feature encoder and the first K Transformer layers.

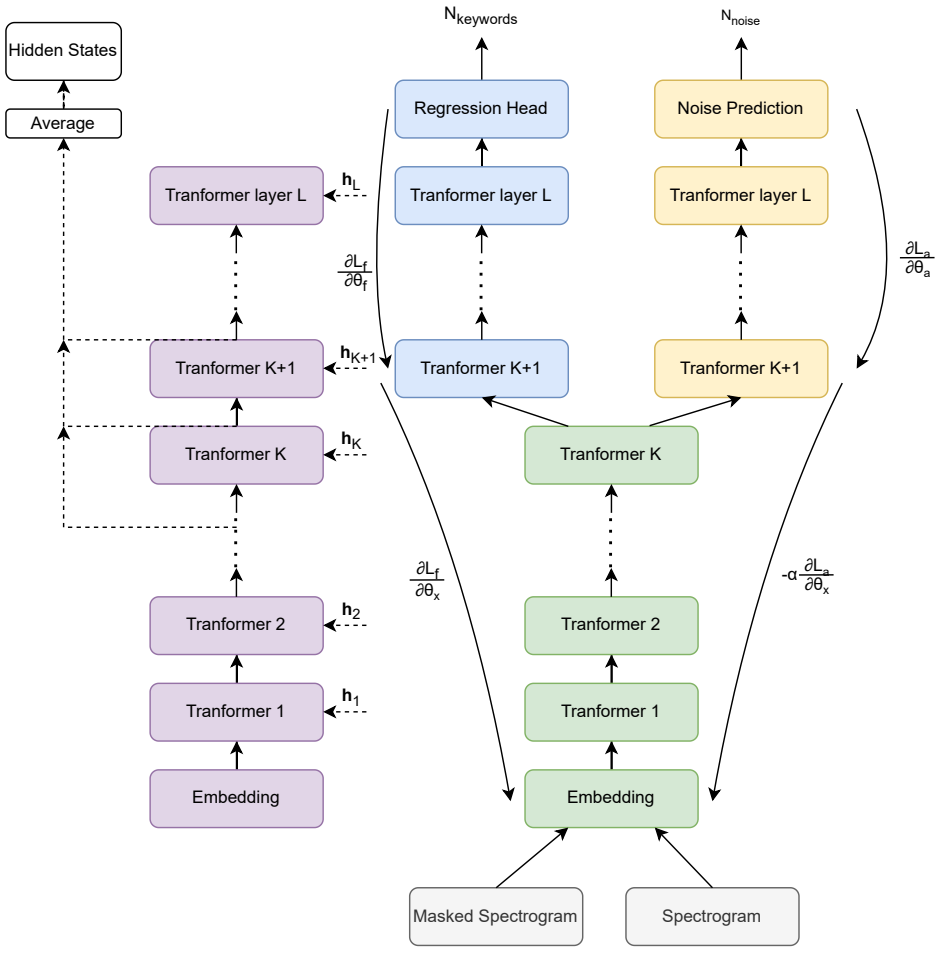

Figure 5.1: Structure of the adversarial pertaining model. The shared layers are depicted with green, the friendly student layers with blue, the teacher model with purple, and the adversarial model with yellow.

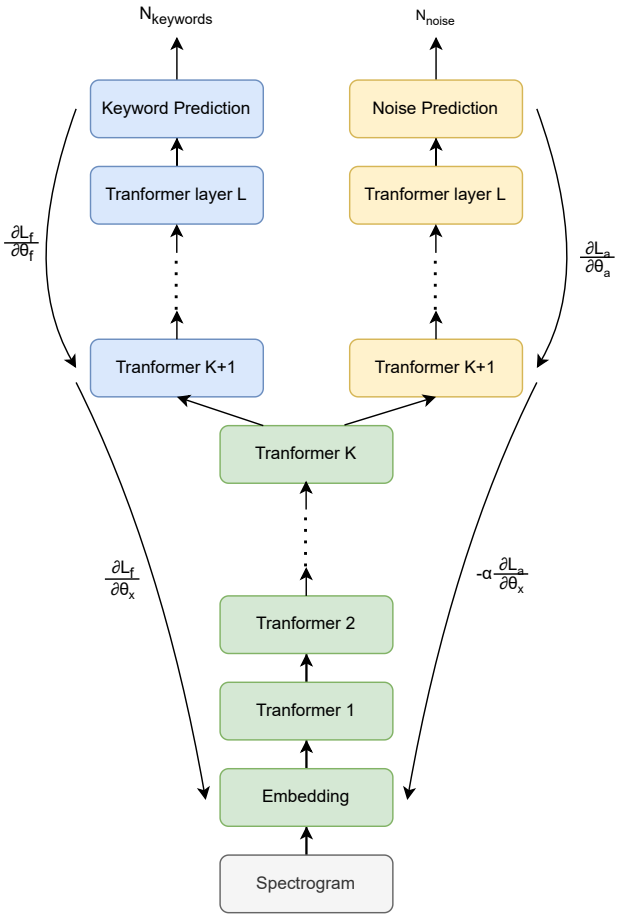

Figure 5.2: Structure of the adversarial finetuning model. The shared layers are depicted in green, the friendly model is blue, and the adversarial model is yellow.

The friendly and adversarial KWTs are optimized over the cross-entropy loss functions $\mathcal{L}_f$ and $\mathcal{L}_f$, respectively, while the shared layers are optimized over both $\mathcal{L}_f$ and $\mathcal{L}_a$. The signs of the gradients calculated from $\mathcal{L}_a$ are flipped and multiplied with α when updating the weights in the shared layers.

Chapter 6

Methods and Results

6.1 Dataset

6.1.1 Keyword Dataset

The models were trained on the Google Speech Commands Dataset V2 [55] in label-deficient mode. The dataset contains 105,829 utterances of 35 keywords recorded with mobile phone or laptop microphones. The words were spoken both by female and male participants. However, their ratio is unknown. The recordings were done ideally in a closed room, but some keywords are inherently noisy. The recordings are stored in 16 kHz WAV files, and the length of the keywords is restricted to 1 s. The distribution of the keywords is not uniform. Some of the keywords have more than twice as many utterances then others. The number of utterances per keyword is between 1557 and 4052 [55].

The dataset contains a validation and test list. The rest of the keywords belong to the training list. This ensures that the evaluations of KWS systems in other studies were also done on the same training set. To make the label-deficient pretraining dataset, 80% of the training set was removed and added to the pretraining dataset. The labels are not used in the pretraining set. Words were randomly selected to be removed from the training set and to be added to the pretraining set in a way that the distribution of the keywords over the training set and the pretraining set is roughly identical and resembles the original distribution of the keywords over the whole dataset. Table 6.1 shows the number of utterances in the pretraining, training, validation, and test sets.

Dataset	Number of Utterances
Pretraining	67874
Training	16969
Validation	9981
Testing	11005

Table 6.1: Number of utterances in each set

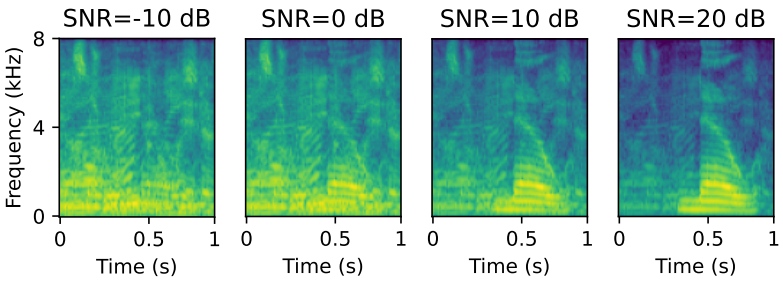

Figure 6.1: Consistent BBL background noise for the keyword *"No"*

6.1.2 Noise Dataset

Six types of noise are used in the experiments. The noise types of *bus* (BUS), *cafe* (CAF), *pedestrian* (PED) and *street* (STR) were used from CHiME-3 [3]. The noise types of *babble* (BBL) and *speech-shaped noise* (SSN) were generated by the authors of [33]. Each noise contains 40 minutes of noise audio for training data (except SSN, which contains 50 minutes) and 5 minutes for validation and testing. Some noise types are rather stationary, for example, BUS, PED, STR, and SSN, while others are non-stationary, such as CAF or BBL. Non-stationary noise is known to be more difficult to deal with. All noises are stored in 16 kHz WAV files.

For adding noise to the training, validation, and test data, Filter and Noise Adding Tool (FaNT) [23] was used. FaNT makes it possible to filter speech audio or add noise with different signal-to-noise ratio levels. In this application, the latter was used. The noise was added to the utterances with different SNR levels, but all were between -10 dB and 20 dB.

To add noise to the training test and validation data, a random position in the appropriate noise audio was chosen, and the subsequent 1 second was mixed with the keyword audio. Each position was chosen randomly, but all positions of the noise are identical in different signal-to-noise ratio levels by setting up a seed parameter for every keyword. That was done in order to keep the noise generation consistent because some parts of the noise file could put the network against greater challenge than others. With this technique, it is ensured that the difference is only in the SNR levels.

6.2 Computation Resources

The models were implemented in Python 3.8. The MFCC spectrograms were extracted using the Librosa package. PyTorch 1.13.1 was used for the development of the neural network models. PyTorch is a popular deep-learning library that supports NVIDIA's CUDA toolkit, which enables the running of neural networks on a Graphical Processing Unit (GPU).

With 5.4M parameters of the largest KWT model, the trained models are relatively small compared to the models of other studies, which makes the model able to train and run on a personal computer. In fact, Berg et al. ran the KWTs on a OnePlus 6 mobile phone device using TensorFlow light [4].

Nevertheless, the models were trained and tested on Aalborg University's CLAAUDIA Strato server. The instance *gpu.t4-large* was used, which has 10 virtual cores of Central Processing Units (CPUs), 40.04 GB Random Access Memory (RAM), and an NVIDIA Tesla T4 GPU with 16 GB of Graphics Double Data Rate 6 (GDDR6) video RAM (VRAM). The instance is based on an Ubuntu 18.04 image.

CLAAUDIA has another server for deep learning called AI Cloud which is based on two NVIDIA GDX-2 with Tesla V100 GPUs. AI Cloud has more computational performance than Strato. However, it is a very busy server with a waiting list to write machine learning jobs. For that reason, it is more flexible and faster to run smaller deep-learning models on Strato.

The deep learning processes were tracked with Weights and Biases (WandB). WandB allows the user to track the learning procedures without directly accessing the server, as well as graphically analyze the model during training. This makes recognizing early anomalies possible.

For the experiments, more than 200 models were trained, and more than 1000 tests were conducted. The simplest KWT-1 model takes around 20 minutes to train. The pretraining of the KWT-3 takes roughly 10 hours to train. The adversarial models in section 6.6 took twice as long as normal models, and to achieve figures like figure 6.12, several days of training were required.

The files for the project are publicly available at `https://github.com/GregTheHunInDk/Robust_KWT.git`

6.3 Baseline

In the first step, the baseline model is acquired by pretraining and finetuning the KWT models on the clean dataset. The KWT models are adopted from Bovbjerg's project. All the models are randomly initialized with a seed value, which ensures that the model's initial parameters of the model are the same at the beginning of every training. However, the pretraining-training dataset split is done randomly and differs every time it is run. That means that slight alteration is noticeable in the baseline results. For this reason, the baseline results are redefined for precise comparisons.

The models were studied in two modes: with and without pretraining. This allows us to observe how self-supervised pretraining affects the overall performance of the model.

The model is trained on the MFCC spectrograms of the input audio where the Fast Fourier Transform (FFT) resolution was set to 480, and the length of the sliding window was 480 samples with a hop length of 160. The first 40 MFCC features were used. For data augmentation, SpecAugment was employed during training for data augmentation, which allowed the model to learn richer features.

Model	Accuracy
KWT-1	0.8488
KWT-2	0.8602
KWT-3	0.8343

Table 6.2: KWT accuracies after being trained on the reduced training dataset

6.3.1 Baseline Without Pretraining

First, the models were trained on the reduced training dataset in supervised mode. The models were trained for 140 epochs with a batch size of 512. The target labels were smoothed with $\epsilon = 0.1$. The weights were optimized with AdamW optimizer with a maximum learning rate of 0.001, which was warmed up linearly over the first 10 epochs and then annealed with a cosine annealing schedule. The weight decay of $\lambda = 0.1$ was used to prevent the model from overfitting.

The model consists of 12 blocks of Transformers with attentions heads of 1, 2, and 3 for KWT-1, KWT-2, and KWT-3, respectively. In this order, the input dimensions hence become 64, 128, 198 so that the $d/k = 64$ is kept, where d is the input dimension and k is the number of attention heads.

The baseline accuracies are shown in table 6.2. In this experiment, KWT-2 had the best accuracy with 0.8602, followed by KWT-1 with 0.8488, and the largest model, KWT-3, with 0.8343. The results are slightly different compared to the ones published by Bovbjerg et al. [6], which again is due to the slight change in the pretraining and training dataset.

6.3.2 Baseline with Pretraining

The models are pretrained in the data2vec system on the label-deficient pretraining dataset. The models are trained for 200 epochs and with a batch size of 512. The weights are optimized over an MSE loss function with AdamW optimizers. The top learning rate was 5×10^{-4}, and it was scheduled using a 1-Cycle learning rate scheduler. To prevent overfitting, a weight decay of $\epsilon = 0.1$ was used. The hyperparameters of the student and teacher Transformer networks are the same as the appropriate KWT model, with the exception that the prediction heads are replaced with regression heads.

The student receives a masked input with a masking probability of 0.65, and then the following ten time steps are masked. During pretraining, the student predicts the average of the top 8 layers of the teacher. The layers of the teacher are normalized with the instance norm, so the model does not have to predict extreme values. The teacher's weights are updated as the EMA of the student weights based on Eq. 3.10, where τ is scheduled from $\tau_l = 0.999$ to $\tau_n = 0.9999$ in 1000 steps, and then kept constant.

After pretraining, the pretrained KWT models are finetuned on the reduced training set with the same hyperparameters as for training the baseline models. Table 6.3 shows the accuracy of the KWT models over the test set after finetuning. It can be seen that data2vec pretraining has a hugely positive effect on the performance of the models. Self-supervised pretraining has the most significant effect on KWT-3 and the smallest on KWT-1 but still increases the accuracy substantially. After finetuning, KWT-2 had the best accuracy of 0.9292, followed by KWT-3 with 0.9194, and at last, KWT-1 with 0.9017. After finetuning, all KWT models perform over 90% on the clean test set.

Model	Accuracy	Improvement over baseline
KWT-1	0.9017	0.0529
KWT-2	0.9292	0.0609
KWT-3	0.9194	0.0852

Table 6.3: KWT accuracies after being pretrained on the label-deficient pretraining dataset and finetuned on the reduced training dataset

6.4 Noise Robustness of Baseline Models

To test the robustness, the Keyword Transformer models were tested with a noise-contaminated test dataset. The background noise BBL, BUS, CAF, PED, SSN, and STR were added to the noise dataset independently with SNR $\in \{-10, -5, 0, 5, 10, 15, 20\}$ dB. That means that 42 test datasets were created, where one dataset contained one type of noise with one signal-to-noise ratio.

Figure 6.2 shows the results of KWT-1 on noisy tests for baseline (i.e., no pertaining) and finetune models. It can be seen that BBL and SSN have the greatest impact on the model on higher noise levels when the SNR $\leq$ 0 dB. STR and BUS, on the other hand, have the least contribution to the decrease in performance. It can be also seen that pretraining has a positive effect on the robustness, with the biggest performance boost on SSN at SNR = 5 dB. At KWT-2 (figure 6.3) and KWT-3 (figure 6.4), the same pattern can be recognized with the subtle difference that the biggest performance gain was found in STR noise at 0 dB SNR.

In order to visualize the difference between models, an element-wise subtraction was applied between the results of the KWT models, which can be seen in Figure 6.5 In the baseline models, KWT-2 has a slightly better performance in some areas than KWT-1 4% maximum increase, however in some areas for high noise level the performance slightly decreases with a maximum of 2%. KWT-3 performs worse than KWT-1 and KWT-2, which compares to its baseline accuracy in clean tests. At 5 dB SSN noise, it has 6% less accuracy than KWT-2 and 3% less than KWT-1 on STR noise at -5dB SNR level.

When finetuned, KWT-2 has consistently and noticeably better performance with a maximum of 8% performance increase than KWT-1 at 0dB SNR STR noise and 4% performance increase over KWT-3 at SSN 5 dB SNR. KWT-3 has an overall better performance than KWT-1, with a maximum rise of 7% with STR noise at -5 dB SNR. The finetuned differences are shown in figure 6.6.

The experiment shows that pretraining KWT models on clean data are beneficial to increase noise robustness. In fact, in some cases, when the models are finetuned, the performance gain can exceed 20%, which is more than twice the performance gain on clean data.

6.5 Multi-Style Training

Several studies have shown that multi-style training, i.e., training with noisy data, can have a positive effect on the robustness of the ASR tasks, including keyword posting [15]. This section will study how adding noise can affect the robustness of a Keyword Transformer. In supervised training, the core idea is to make the model learn noise-robust discriminative features that it can classify over. Simple self-supervised pretraining on noisy data can be even more harmful because the model learns fewer speech and more noise features. For that reason, the noise will only be added to the finetuning dataset, and it will be studied whether pretraining on clean data can help the model learn noise-independent features.

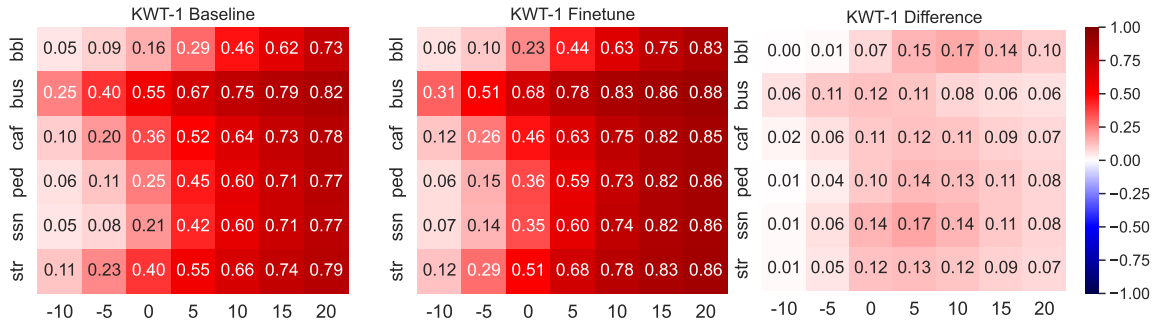

Figure 6.2: Noisy test results for KWT-1.

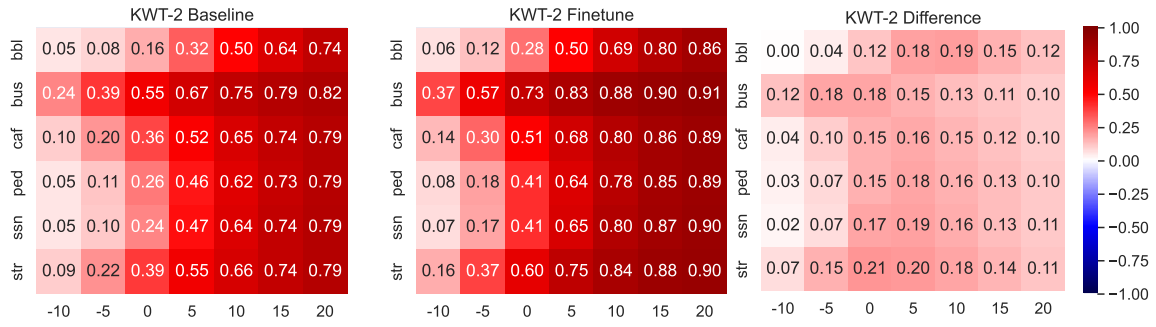

Figure 6.3: Noisy test results for KWT-2.

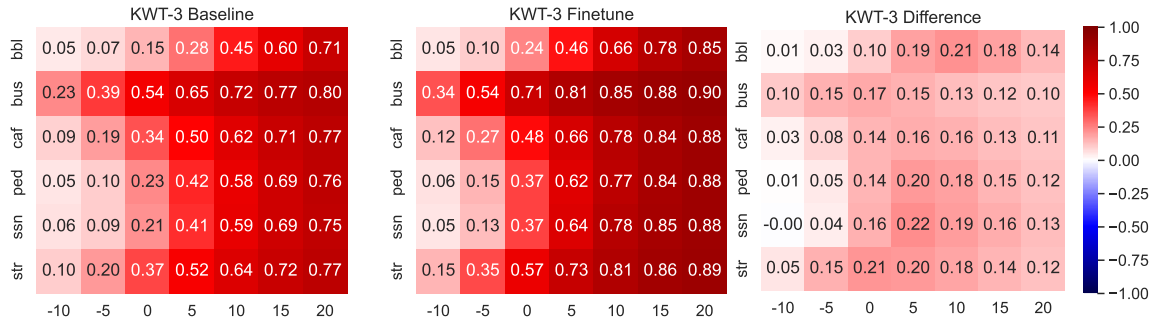

Figure 6.4: Noisy test results for KWT-3.

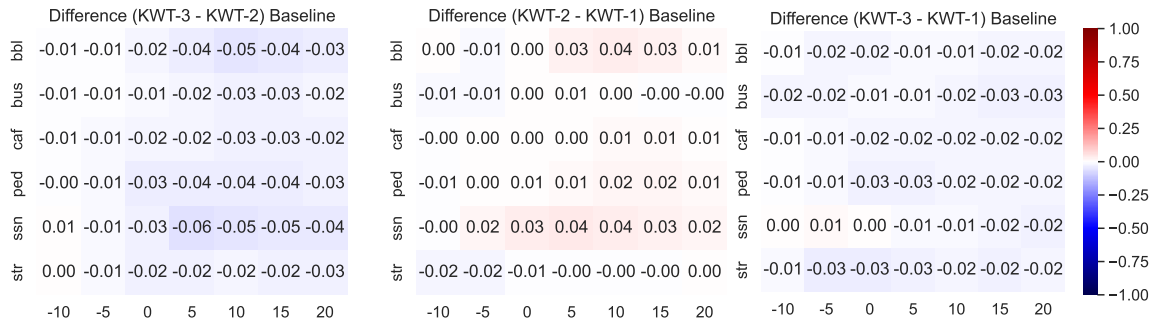

Figure 6.5: Difference between the accuracies of the KWT baseline models

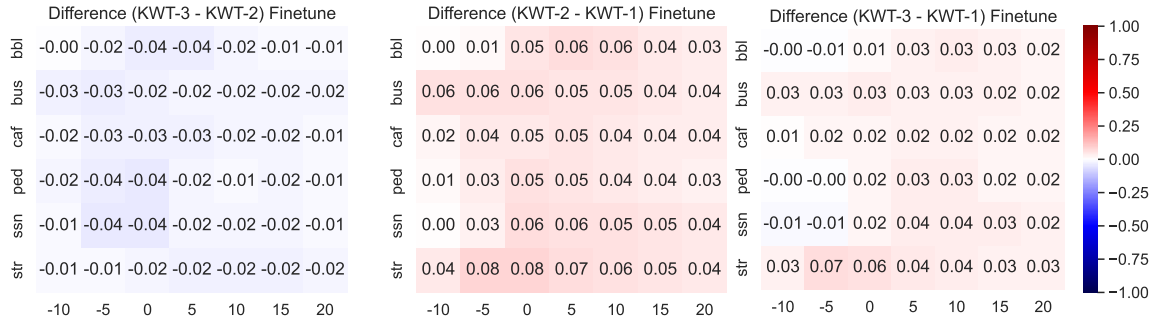

Figure 6.6: Difference between the accuracies of the finetuned KWT baseline models.

In this experiment, stationary and non-stationary noises were added to the reduced training data. Since in [37], it was found that adding only one type of noise to the training dataset could lead to better results than adding a mixture of noises, this experiment also follows that strategy. For noisy training, two noise types are chosen, one stationary and one non-stationary, which are SSN and BBL. Both noises are added to the training, validation, and test sets with SNR $\in \{-10, -5, 0, 5, 10, 20\}$.

The six models per noise type, one for each SNR level, are trained with the same hyperparameters as the baseline. After that, all models were tested with all the SNR values. The models were tested with the type of noise they were trained on. For example, one model was trained on SSN-noisy data with an SNR of 0 dB. Then the model was tested with test data contaminated with SSN of SNR $\in \{-10, -5, 0, 5, 10, 15, 20\}$.

In this section, the results for KWT-3 are presented and compared to KWT-2. Noisy training behaves similarly to KWT-1, but during the training, some of the data became corrupt, and for that reason, KWT-1 cannot bring a perfect picture of finetuning. Moreover, in this section, only results with SSN data are presented, but the models behave very similarly on BBL noise. Results with BBL noise are presented in the appendix A.1.

Effect of Pretraining

First, KWT-3 is trained on SSN data without any pretraining. Then, clean pretraining and SSN-noisy finetuning are applied. Figure 6.7 shows the results. The performance of the model decreases drastically with the increase of test SNR. The smaller the training SNR is, the more intense differ-

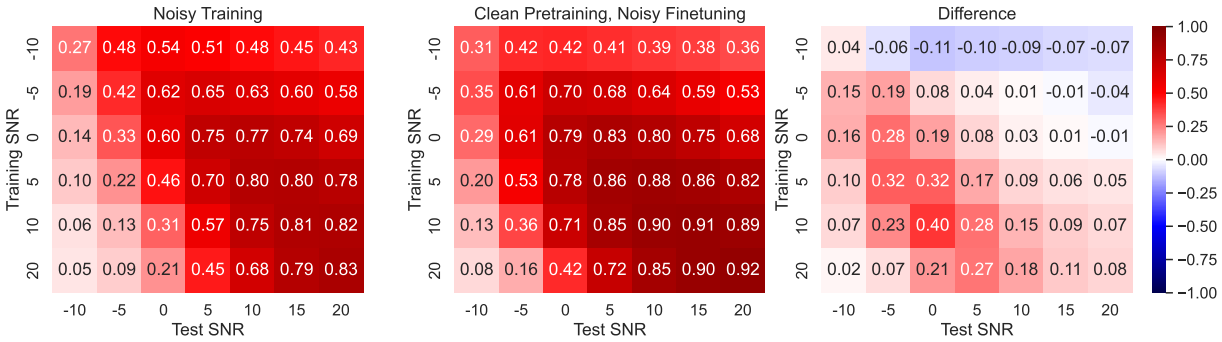

Figure 6.7: Noisy test results for KWT-3. On the first heatmap, the results without pertaining are depicted. On the second heatmap, the KWT-3 models are pretrained on clean data and finetuned on noisy. On the third heatmap, the improvement by pretraining can be seen.

ence can be seen in the different test cases. For training SNR=20dB, the model performed almost on the baseline level with 83%, but when the SNR became -10 dB, the accuracy was only 5 dB. On the other hand, the model trained on SNR=-10 dB had the best performance at test SNR=0 dB with 54% accuracy and the worst prediction rate at -10 dB with 27%. The finetuned models have a smoother transition in test accuracies and an overall higher prediction performance as well.

With some exceptions, the noisy trained models have the best performance when the models are trained with slightly less SNR than they are tested on. For the SNR=15 dB test, SNR=10 dB training produces the best results, or for the SNR=10 dB, SNR=5 dB training data gives the best performance. When finetuned, the test accuracy peaks when the test SNR and training SNR are roughly identical.

The last heatmap of figure 6.7 shows that clean pretraining mostly increases the model accuracy when finetuned on a noisy set, compared to training on noisy data without any pretraining. The larger performance gain is experienced at SNR=0 dB, at 10 dB training SNR with a huge 40% increase. On the other hand, when the training SNR is low, especially at -10 dB, then clean pretraining has a negative effect on the performance. The greatest performance decrease is -11% at 0dB test and -10 dB training SNR.

Effect of Model Size

To observe how model size affects affect the performance of the Keyword Transformer after noisy training, KWT-3 and KWT-2 are compared both for noisy training and clean pretraining - noisy finetuning modes. Figures 6.8 show the difference between KWT-3 and KWT-2 test performances. The values are received by an elementwise subtraction of the KWT-3 and KWT-2 test results.

In a noisy training-only system, KWT-3 performs better on low-SNR training and high-SNR test cases. At -10 dB training SNR and 0 dB testing SNR, KWT-3 performs 22% better than KWt-2. On the contrary, KWT-2 performs better on high-SNR training and low SNR-test cases with a maximum increase of 20% over KWT-3 at 10 dB training and 0 dB testing SNR.

When finetuned, KWT-3 performs marginally better than KWT-2 with a maximum increase of 4% at 5 dB training and -5 dB testing SNR and for -5 dB training and 20 db testing SNR cases. On the other hand, KWT-2 performs better at low-SNR training and low-SNR test case with a maximum 3% increase at -5 dB training and -5 dB SNR testing cases.

The experiments show that KWT-2 and KWT-3 learn noise-robust speech representations equally

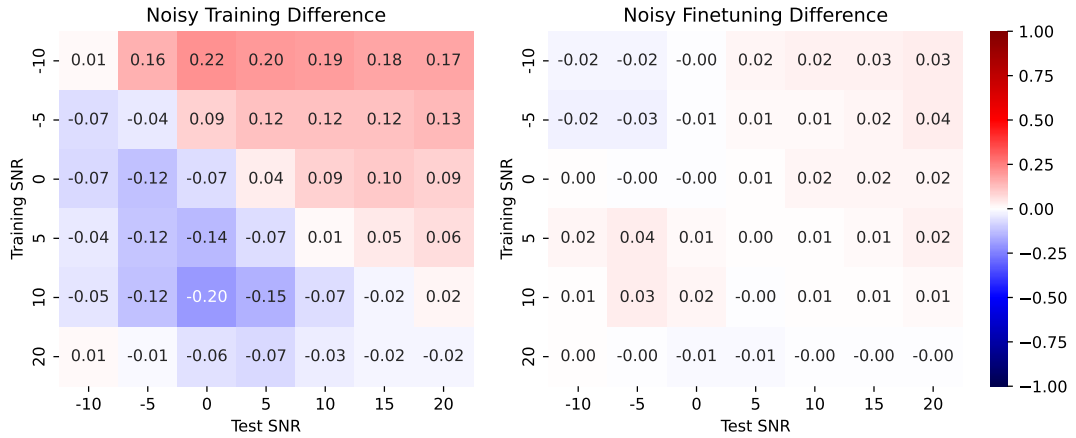

Figure 6.8: Difference between KWT-3 and KWT-2 performance for noisy training (left) and clean pertaining - noisy finetuning (right) modes.

Figure 6.9: Difference between noisy training and clean training for KWT-3

well when finetuned. KWT-2, on the other hand, has the advantage of having fewer parameters while giving a competitive or even better performance than KWT-3.

Comparing to Clean Training

To see in what cases noisy training helps increase noise robustness, the noisy-trained KWT-3 model is compared to a clean-trained KWT-3 model. Figure 6.9 shows results. The left heatmap shows the test accuracy after noisy training. The middle heatmap shows the test accuracy for a clean-trained KWT-3. The right heatmap shows the difference in the accuracies. KWT-3 mostly increases the robustness in a purely noisy environment. The biggest boost can be experienced at low-SNR training and test conditions, but noisy training generally increases the performance when the training and testing SNRs are roughly equal. The only part where noisy training has a negative effect is in low-SNR training and high-SNR testing cases. The biggest performance gain is 40% at -5 dB training and 0 dB testing SNR, and a large accuracy loss is -32% at -10 dB training and 20 dB testing SNR.

When finetuning the KWT-3 models on noisy and clean data, the same results with more contrast can be observed. Noisy-finetuned KWT-3 has much higher accuracy than the clean-finetuned one, with a maximum gain of 49% at 0dB training and -5 dB testing SNR. On the other hand, the performance loss is also more severe in low-SNR training and high-SNR testing cases, with a

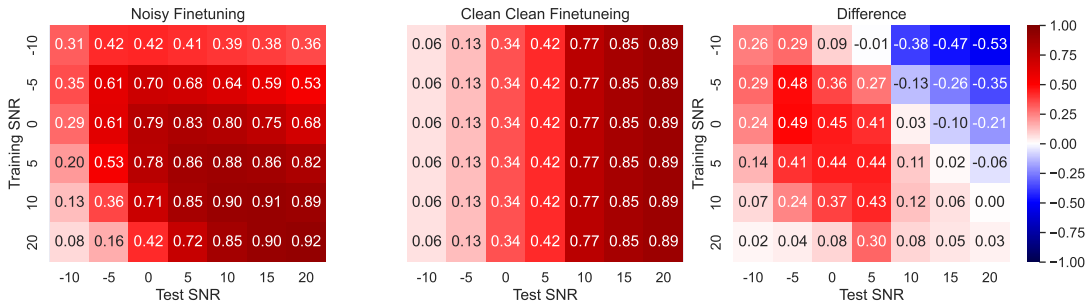

Figure 6.10: Difference between noisy finetuning and clean finetuning for KWT-3

maximum of -53% performance loss. In noisy cases, however, the overall performance increases when the model is finetuned to noisy data.

In conclusion, in noise-specific cases, training on noisy data overall helps increase the robustness of keyword spotting. If the signal-to-noise ratio is known in practical applications, it is advised to train the noise with similar SNR levels. On the other hand, finetuning the model on clean data can further increase the noise robustness of the Keyword Transformers. The model size, on the other hand, does not necessarily increase the noise performance, and for that reason using smaller models can be beneficial in practical applications.

6.6 Adversarial Training

Adversarial training can be applied both on pretraining and finetuning. In the experiments, two main methods were used in adversarial training. In the first method, following the VAD study [34], both the adversarial and friendly networks receive noisy data. In this project, this method is called noisy adversarial training. In the second method, the friendly network receives clean audio, and the adversarial task receives the noisy equivalent of the keywords. This method was inspired by robust data2vec and enhanced wav2vec2.0, although they used clean and noisy audio on different learning tasks. With an adversarial task included, in pretraining, the network is expected to learn noise-robust speech features, while in finetuning, it is expected to learn to classify over noise-robust speech features.

The experiments were conducted over two parameters, namely α and k. The parameter α controls the rate of adversarial learning, and k denotes the bottom K Transformer layers that are shared between the friendly and adversarial models. The parameters α and k were chosen from the sets $\alpha \in \{0.1, 0.01\}$ and $k \in \{1, 2, 3, 4, 5, 6\}$.

All other model hyperparameters are identical to the baseline models to keep the comparisons consistent. That means that these models are pretrained and finetuned for 200 and 140 epochs with a batch size of 512. All the other hyperparameters are also the same, including the learning rate, the scheduling, the label smoothing, etc.

During the training, at least the adversarial network need noise-contaminated data. Following the VAD study, the training data was mixed with four types of noise: BBL, BUS, CAF, STR, and CLN, where CLN means, that the data was kept clean. 80% of the training data was contaminated with noise, where the distribution of the noise types was kept uniform. During testing, clean data was omitted, so the 100% test data was mixed with BBL, BUS, CAF, and STR with a uniform distribution. The signal-to-noise ratio was chosen between 0 dB and 10 dB.

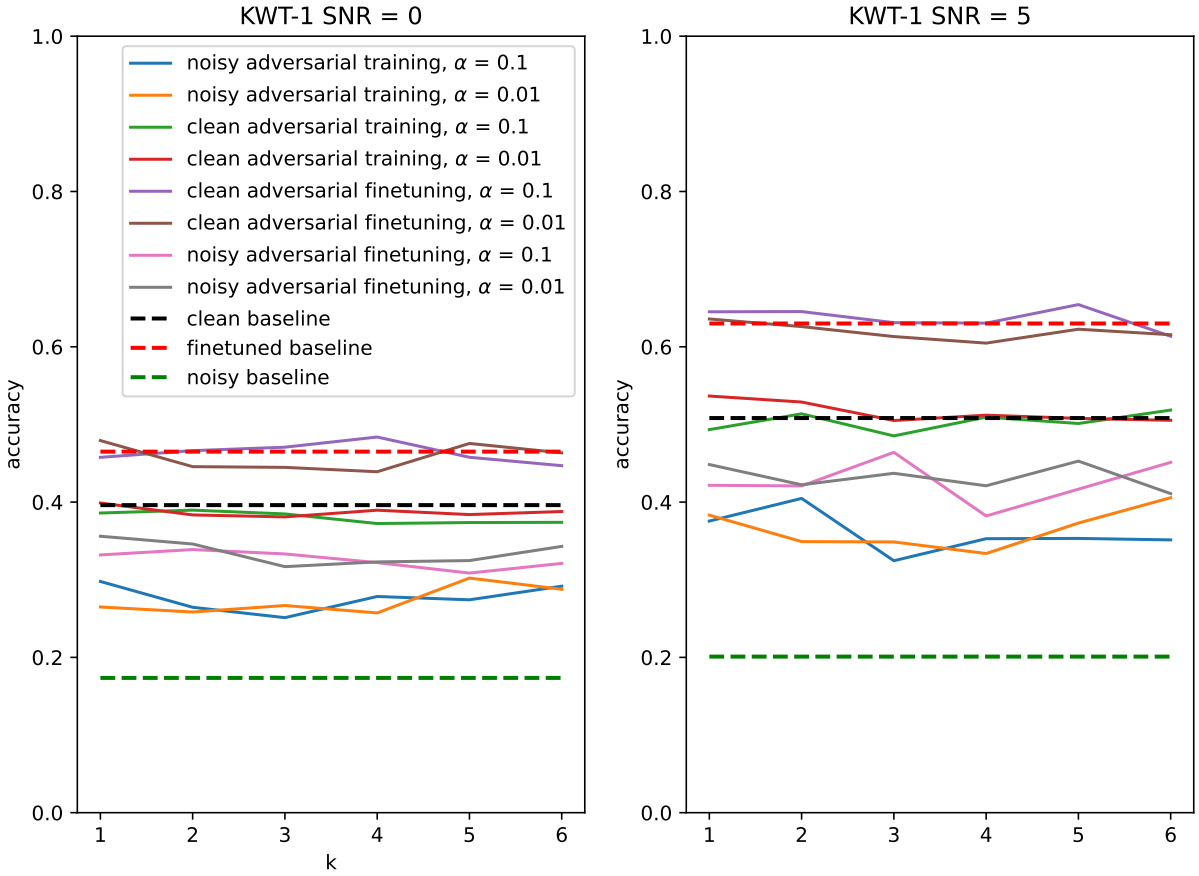

Figure 6.11: Test results of adversarial models for SNR=0 and SNR=5. The legend shown on the left side applies to both plots.

6.6.1 Supervised Adversarial Training

Varying α and k

The first experiment studies how varying α and k can affect the prediction performance. Considering its shortest training time, KWT-1 is chosen for adversarial training. The training is done in three modes: noisy adversarial training (when both the friendly and adversarial network receives noisy data), clean adversarial training (when only the adversarial network receives noisy data), clean adversarial finetuning, and noisy adversarial finetuning. The value of α is first set to 0.1, and the k is iterated from 1 to 6. After that, the parameter α is then set to 0.01, and the models are trained and tested for all k from 1 to 6. The adversarial models are then compared with baseline values, both with and without pretraining. In total, 4 training configurations with 6 values of k and 2 values of α were trained and tested, with an additional clean data2vec pertaining, which makes it a total of 48 adversarial training, 1 clean pertaining, and 2 finetuning per SNR.

Figure 6.11 shows the testing results on the adversarial networks for SNR=0 dB and SNR=5 dB. On the horizontal axis, the changing of the number of the first K Transformer layers is depicted. The accuracy is shown on the horizontal axis. The line styles and colors belong to the same training

Training type	α	Mean accuracy	Maximum accuracy
Noisy adversarial training	0.1	0.2763	0.2978
Noisy adversarial training	0.01	0.2729	0.3021
Clean adversarial training	0.1	0.3800	0.3896
Clean adversarial training	0.01	0.3874	0.3988
Noisy adversarial finetuning	0.1	0.3259	0.3388
Noisy adversarial finetuning	0.01	0.3349	0.3560
Clean adversarial finetuning	0.1	0.4638	0.4838
Clean adversarial finetuning	0.01	0.4579	0.4792
Clean training	-	0.3960	0.3960
Noisy training	-	0.1736	0.1736
Clean finetune	-	0.4650	0.4650

Table 6.4: Adversarial training results for SNR=0

Training type	α	Mean accuracy	Maximum accuracy
Noisy adversarial training	0.1	0.3604	0.4047
Noisy adversarial training	0.01	0.3656	0.4056
Clean adversarial training	0.1	0.5036	0.5185
Clean adversarial training	0.01	0.5159	0.5366
Noisy adversarial finetuning	0.1	0.4261	0.4641
Noisy adversarial finetuning	0.01	0.4321	0.4528
Clean adversarial finetuning	0.1	0.6366	0.6543
Clean adversarial finetuning	0.01	0.6196	0.6358
Clean training	-	0.5082	0.5082
Noisy training	-	0.2010	0.2010
Clean finetune	-	0.6300	0.6300

Table 6.5: Adversarial training results for SNR=5

type, for example, the blue line means noisy adversarial training with $\alpha = 0.1$ for both SNR=0 and SNR=5.

Comparing clean and noisy adversarial training, clean adversarial training produces better accuracy than noisy ones for all α and k. At SNR=0, adversarial clean training produces slightly worse results than the baseline with 38% and 38.74% for $\alpha = 0.1$ and $\alpha = 0.01$ compared to the baseline accuracy of 3960%. At SNR=5 dB, adversarial training manages to improve over the baseline results for $\alpha = 0.01$ with a maximum increase of 2.84%. Noisy adversarial training, on the other hand, produces significantly worse results than the baseline with 36.04% and 36.56% for $\alpha = 0.1$ and $\alpha = 0.01$, respectively. Nevertheless, all adversarial training performs better than training the model on a noisy dataset. In the previous section, it was mentioned that training Keyword Transformers on noisy data can increase the accuracy of the model in noise-specific cases. However, this particular mix of noise makes the model fail to learn to generalize keywords adequately.

Next, the models are pretrained on clean data and finetuned in the adversarial setup. Pretraining visibly increases the accuracy of clean and noisy adversarial training. However, noisy adversarial finetuning still produces worse results than the baseline. Clean adversarial fine-tuning with $\alpha = 0.01$ at SNR=5 dB, on the other hand, produces better mean accuracy than a KWT-1 model pretrained and finetuned on clean data. the maximum performance gain is 2.43% at $k = 5$. All other models manage to get better performance for certain k values. For SNR=5 and $\alpha = 0.1$, the peak performance is 63.58% at $k = 1$. At SNR=0 dB, the peak accuracy increase for $\alpha = 0.1$ is 01.88% at $k = 4$, and for $\alpha = 0.01$, the maximum increase is 1.42%.

The results show that robustness can be increased with adversarial training when the friendly model receives clean data, especially when the signal-to-noise ratio is higher. There is, on the other hand, no clear correlation found between the accuracies and the value of α and k.

Varying the model size

To see how model size can affect the adversarial training outcomes, the SNR is fixed to 0 dB, as well as α to 0.1, while k is iterated from 1 to 6, and the experiments are repeated for KWT-1, KWT-2, and KWT-3. The models are trained in four modes: adversarial noisy training, adversarial noisy finetuning, adversarial clean training, and adversarial clean finetuning. The models are trained with the baseline models trained on clean data and tested on noisy data. The training, validation, and test datasets are mixed with noises in the same way as in the section 6.6.1. Figure 6.12 shows the results for KWT-1, KWT-2, and KWT-3.

The performance of the network on adversarial training decrease with larger model sizes, which means, that KWT-2 and KWT-3 react more on data augmentation than KWT-1, unfortunately, for the worse. Since it is known from the previous experiments of section 6.6.1 that training the KWT-1 on data contaminated with this particular mix of noise has a negative effect on the test performance, these results are not surprising.

Comparing KWT-2 to KWT-3, KWT-2 performs better in every aspect but, nevertheless, below the baseline models. Similarly to section 6.6.1, no clear dependence can be seen between the accuracy and the value of k. For certain k values, KWT-3 can reach higher values, but the best performance and mean performance of KWT-2 are always higher.

6.6.2 Semi-Supervised Adversarial Pretraining

In self-supervised pertaining, the model is expected to learn noise-robust speech representations by learning clean speech representations in the friendly task while unlearning noise representations.

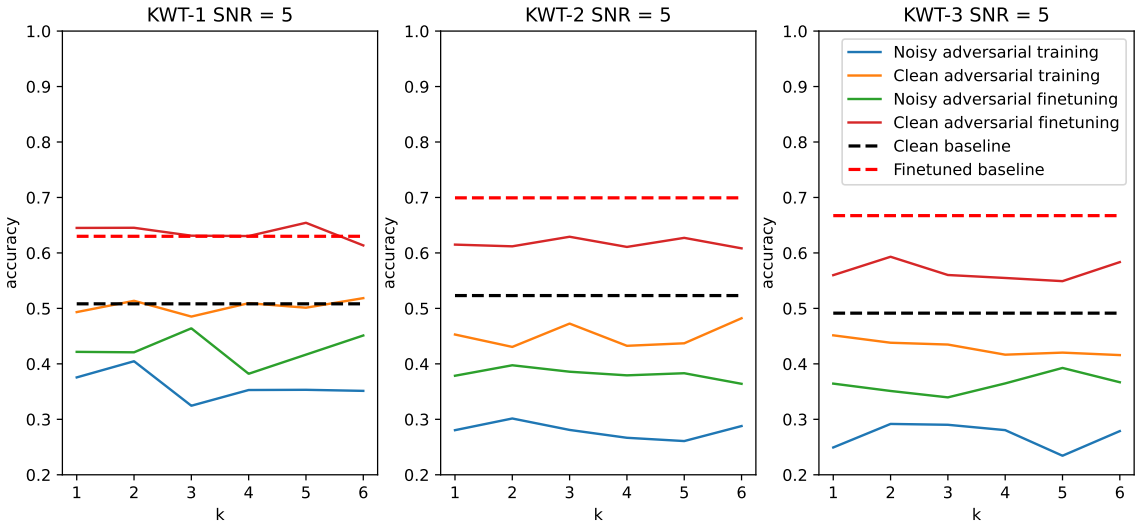

Figure 6.12: Adversarial training results for KWT-1, KWT-2 and KWT-3 for $\alpha = 0.1$ and SNR=5 dB

Training type	Mean accuracy	Maximum accuracy
KWT-1		
Noisy adversarial training	0.3604	0.4047
Noisy adversarial finetuning	0.4261	0.4641
Clean adversarial training	0.5036	0.5185
Clean adversarial finetuning	0.6366	0.6543
KWT-2		
Noisy adversarial training	0.2797	0.3015
Noisy adversarial finetuning	0.3814	0.3975
Clean adversarial training	0.4513	0.4822
Clean adversarial finetuning	0.6171	0.6292
KWT-3		
Noisy adversarial training	0.2708	0.2917
Noisy adversarial finetuning	0.3632	0.3926
Clean adversarial training	0.4295	0.4514
Clean adversarial finetuning	0.5668	0.5931

Table 6.6: KWT-1, KWT-2 and KWT-3 results with adversarial training for SNR=5 dB and $\alpha = 0.1$

Training method	Accuracy	
	Clean	Noisy
Clean training	0.8488	0.3960
Clean pretraining, clean finetuning	0.9017	0.4750
Adversarial pretaining, clean finetuning	0.8617	0.4006

Table 6.7: Adversarial pretrained and clean finetuned in comparison with clean training and clean pertaining and finetuning

Self-supervised pretraining is the longest process in the experiments for being trained on the four times larger pretraining dataset and having two networks, a student and a teacher. Adding the parallel adversarial task makes the pretraining procedure even longer. For that reason, the pretraining experiment is simplified by relying on the following facts learned from sections 6.6.1 and 6.6.1: there is no strong correlation between the number of shared layers K, the learning parameter α and the model accuracy; moreover, the model behaves similarly at larger signal-to-noise ratios and at lower ones. Section 6.6.1 showed that adversarial pretraining works better for KWT-1 than for KWT-2 and KWT-3.

Considering all the above-mentioned facts, to conduct the experiments, KWT-1 was chosen, with $k = 1$ and $\alpha = 1$. The training data is mixed with BBL, BUS, CAF, and PED, and the ratio of the noisy data to the clean one is 4:1. The test data is mixed with BBL, BUS, CAF, and PED as well, but no data was kept clean, to see, how the model behaves in a purely noisy environment. The signal-to-noise ratio is kept at 0 dB both for training and testing. In order to observe the effect purely of the adversarial pretraining, the model is finetuned on the clean dataset.

Table 6.7 shows the results after adversarial pretraining and clean finetuning. Adversarial pretraining with clean finetuning performs better than the baseline model simply trained on the reduced dataset. However, a fair comparison is between the model pretrained on clean data versus the model pretrained in an adversarial manner. The clean pretrained KWT-1 model performs much better in clean and noisy scenarios, 6.17% better on the clean test and 7.44% on the noisy test.

This experiment shows that with adversarial pertaining, the KWT-1 model learns weaker representations of human speech. It still performs better than the baseline without pertaining, but clean pretraining gives the model a much better gain in noise robustness.

Chapter 7

Conclusion

In this master thesis, the acoustical noise robustness of Keyword Transformers was studied with the aim of enhancing the robustness using supervised and self-supervised learning. This study focused on back-end methods to increase noise robustness while keeping the original feature extractor. While other keyword-spotting studies mostly focus on the clean performance of KWS systems, this study focused on increasing the noise robustness and keeping the prediction accuracies as close to the baseline as possible.

In the thesis, first, the background of the project was explained, and the research question was formed. After that, the keyword spotting systems were introduced, as well as common methods to increase their noise-robustness were explained. Following that, the baseline models were presented that the project was based on. Then other works were shortly presented that served as inspiration for the methods applied to Keyword Transformers. Later the adopted Keyword Transformer models were explained, as well as the applications of multi-style and adversarial learning applied on them. Lastly, the experimental methods and results were presented in detail.

By studying the robustness of the Keyword Transformer models, it was clearly shown that pretraining the Transformers in the data2vec architecture not only increases the performance of the models on clean data but also in the presence of noise. In multi-style learning, it was shown that the model performed better in noisy test cases when the training data was mixed with noise. It also shows that model size does not play a key role in the robustness. However, pretraining the model on clean speech does. Finally, it was also shown that introducing noise in the training phase has an overall positive effect when the model is expected to be used in a noisy environment.

In the adversarial training study, it was shown that the models have marginal performance gain at best as well as introducing mixed noise in the friendly task has a negative effect on the test accuracy. It was also demonstrated that varying the value of the learning rate as well as the number of shared Transformer layers does not have a direct effect on the accuracy. It was also manifested that adversarial training affects larger Transformer models stronger. Finally, adversarial pretraining was shown that introducing adversarial tasks in the pretraining had a negative effect on learning speech representations.

Based on the results, the research question can be answered, which was formed as follows:

What supervised and self-supervised training methods can be used to enhance the noise-robustness of the Keyword Transformer?

The experiments showed that among the tried methods, clean self-supervised training is the best to increase the noise robustness of the Keyword Transformer. It was also shown that pretraining on clean data has almost always a positive effect on robustness, no matter what finetuning technique is combined with it.

In supervised training, introducing one type of noise during the training has an overall positive effect on the robustness, while adding a mixture of different noises to the training set can decrease the robustness severely. It was also shown that adversarial training works better with smaller KWT models, and although the mixture of noise decreased the noise robustness, supervised adversarial clean training could increase the performance of the model.

In this study, the adversarial training was done with a mixture of noise, although it was shown that simply training the KWT models with mixed-noisy data reduces the performance. On the other hand, adding only one type of noise to the data and training the models with it showed great performance gain in the noisy test. In the future adversarial training could be executed using only one type of noise in the hope of increasing noise robustness in a particular noisy case.

To conclude, the robustness of the Keyword Transformers can be increased by introducing noisy data to the supervised training set, adding adversarial learning to the supervised task, or pretraining the network on clean data in a self-supervised mode.

Bibliography

[1] Alexei Baevski et al. "data2vec: A General Framework for Self-supervised Learning in Speech, Vision and Language". In: *Proceedings of the 39th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri et al. Vol. 162. Proceedings of Machine Learning Research. PMLR, 2022, pp. 1298–1312. URL: https://proceedings.mlr.press/v162/baevski22a.html.

[2] Alexei Baevski et al. "wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 12449–12460. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/92d1e1eb1cd6f9fba3227870bb6d7f07-Paper.pdf.

[3] Jon Barker et al. "The third 'CHiME' Speech Separation and Recognition Challenge: Dataset, task and baselines". In: *2015 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU 2015)*. Scottsdale, AZ, United States, Dec. 2015. URL: https://inria.hal.science/hal-01211376.

[4] Axel Berg, Mark O'Connor, and Miguel Tairum Cruz. "Keyword Transformer: A Self-Attention Model for Keyword Spotting". eng. In: *Interspeech 2021,Brno, Czech Republic,2021-08-30 - 2021-09-03*. Ithaca: Cornell University Library, arXiv.org, 2021, pp. 4249–.

[5] Holger Severin Bovbjerg. "Self-supervised Keyword Spotting". In: (2022).

[6] Holger Severin Bovbjerg and Zheng-Hua Tan. *Improving Label-Deficient Keyword Spotting Using Self-Supervised Pretraining*. 2022. arXiv: 2210.01703 [cs.SD].

[7] Guoguo Chen, Carolina Parada, and Georg Heigold. "Small-footprint keyword spotting using deep neural networks". In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2014, pp. 4087–4091. DOI: 10.1109/ICASSP.2014.6854370.

[8] Guoguo Chen, Carolina Parada, and Tara N. Sainath. "Query-by-example keyword spotting using long short-term memory networks". In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2015, pp. 5236–5240. DOI: 10.1109/ICASSP.2015.7178970.

[9] Bhusan Chettri, Emmanouil Benetos, and Bob LT Sturm. "Dataset Artefacts in anti-spoofing systems: a case study on the ASVspoof 2017 benchmark". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 28 (2020), pp. 3018–3028.

[10] Seungwoo Choi et al. "Temporal Convolution for Real-time Keyword Spotting on Mobile Devices". eng. In: (2019).

[11] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[12] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV].

[13] "Dynamic Time Warping". In: *Information Retrieval for Music and Motion*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 69–84. ISBN: 978-3-540-74048-3. DOI: 10.1007/978-3-540-74048-3_4. URL: https://doi.org/10.1007/978-3-540-74048-3_4.

[14] Alexandre Défossez et al. *Demucs: Deep Extractor for Music Sources with extra unlabeled data remixed*. 2019. arXiv: 1909.01174 [cs.SD].

[15] Ivan Lopez Espejo et al. "Deep Spoken Keyword Spotting: An Overview". English. In: *IEEE Access* 10 (Jan. 2022), pp. 4169–4199. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3139508.

[16] "Feature Extraction for ASR". In: *Speech and Audio Signal Processing*. John Wiley & Sons, Ltd, 2011. Chap. 22, pp. 301–318. ISBN: 9781118142882. DOI: https://doi-org.zorac.aub.aau.dk/10.1002/9781118142882.ch22. eprint: https://onlinelibrary-wiley-com.zorac.aub.aau.dk/doi/pdf/10.1002/9781118142882.ch22. URL: https://onlinelibrary-wiley-com.zorac.aub.aau.dk/doi/abs/10.1002/9781118142882.ch22.

[17] Yaroslav Ganin et al. "Domain-adversarial training of neural networks". In: *The journal of machine learning research* 17.1 (2016), pp. 2096–2030.

[18] John S Garofolo. "Timit acoustic phonetic continuous speech corpus". In: *Linguistic Data Consortium, 1993* (1993).

[19] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2015. arXiv: 1412.6572 [stat.ML].

[20] Alex Graves et al. "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks". In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 2006, pp. 369376. ISBN: 1595933832. DOI: 10.1145/1143844.1143891. URL: https://doi.org/10.1145/1143844.1143891.

[21] Yue Gu et al. *A Monaural Speech Enhancement Method for Robust Small-Footprint Keyword Spotting*. 2019. arXiv: 1906.08415 [cs.SD].

[22] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. arXiv: 1503.02531 [stat.ML].

[23] H.-G. Hirsch. *FANT: Filtering and Noise Adding Tool Kernel Description*. 2017. URL: https://github.com/i3thuan5/FaNT (visited on 05/30/2023).

[24] Hans-Günter Hirsch and David Pearce. "The Aurora experimental framework for the performance evaluation of speech recognition systems under noisy conditions". In: *ASR2000-Automatic speech recognition: challenges for the new Millenium ISCA tutorial and research workshop (ITRW)*. 2000.

[25] Poul Hoang et al. "The minimum overlap-gap algorithm for speech enhancement". In: *IEEE Access* 10 (2022), pp. 14698–14716.

[26] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[27] Wei-Ning Hsu et al. "Hubert: How Much Can a Bad Teacher Benefit ASR Pre-Training?" In: *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2021, pp. 6533–6537. DOI: 10.1109/ICASSP39728.2021.9414460.

[28] Yiteng Huang et al. "Supervised Noise Reduction for Multichannel Keyword Spotting". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, pp. 5474–5478. DOI: 10.1109/ICASSP.2018.8462346.

[29] Yiteng Arden Huang, Turaj Z. Shabestary, and Alexander Gruenstein. "Hotword Cleaner: Dual-microphone Adaptive Noise Cancellation with Deferred Filter Coefficients for Robust Keyword Spotting". In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019, pp. 6346–6350. DOI: 10.1109/ICASSP.2019.8682682.

[30] Eric Jang, Shixiang Gu, and Ben Poole. *Categorical Reparameterization with Gumbel-Softmax*. 2017. arXiv: 1611.01144 [stat.ML].

[31] Myunghun Jung et al. *Multi-Task Network for Noise-Robust Keyword Spotting and Speaker Verification using CTC-based Soft VAD and Global Query Attention*. 2020. arXiv: 2005.03867 [eess.AS].

[32] Morten Kolbæk, Zheng-Hua Tan, and Jesper Jensen. "Speech intelligibility potential of general and specialized deep neural network based speech enhancement systems". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 25.1 (2016), pp. 153–167.

[33] Morten Kolboek, Zheng-Hua Tan, and Jesper Jensen. "Speech enhancement using long short-term memory based recurrent neural networks for noise robust speaker verification". In: *2016 IEEE spoken language technology workshop (SLT)*. IEEE. 2016, pp. 305–311.

[34] Claus Meyer Larsen, Peter Koch, and Zheng-Hua Tan. *Adversarial Multi-Task Deep Learning for Noise-Robust Voice Activity Detection with Low Algorithmic Delay*. 2022. arXiv: 2207.01691 [eess.AS].

[35] Ilya Loshchilov and Frank Hutter. *Fixing Weight Decay Regularization in Adam*. 2018. URL: https://openreview.net/forum?id=rk6qdGgCZ.

[36] Ilya Loshchilov and Frank Hutter. *SGDR: Stochastic Gradient Descent with Warm Restarts*. 2017. arXiv: 1608.03983 [cs.LG].

[37] Iván López-Espejo, Zheng-Hua Tan, and Jesper Jensen. "A Novel Loss Function and Training Strategy for Noise-Robust Keyword Spotting". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 29 (2021), pp. 2254–2266. DOI: 10.1109/TASLP.2021.3092567.

[38] Iván López-Espejo, Zheng-Hua Tan, and Jesper Jensen. "Improved External Speaker-Robust Keyword Spotting for Hearing Assistive Devices". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 28 (2020), pp. 1233–1247. DOI: 10.1109/TASLP.2020.2984089.

[39] Tobias Menne. "The RWTH/UPB/FORTH system combination for the 4th CHiME challenge evaluation". In: (2016).

[40] Tobias Menne, Ralf Schlüter, and Hermann Ney. "Investigation into Joint Optimization of Single Channel Speech Enhancement and Acoustic Modeling for Robust ASR". In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019, pp. 6660–6664. DOI: 10.1109/ICASSP.2019.8682460.

[41] Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. "When does label smoothing help?" In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/f1748d6b0fd9d439f71450117eba2725-Paper.pdf.

[42] Vassil Panayotov et al. "Librispeech: An ASR corpus based on public domain audio books". In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2015, pp. 5206–5210. DOI: 10.1109/ICASSP.2015.7178964.

[43] Daniel S. Park et al. "SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition". In: *Interspeech 2019*. ISCA, 2019. DOI: 10.21437/interspeech.2019-2680. URL: https://doi.org/10.21437%2Finterspeech.2019-2680.

[44] Niki Parmar et al. "Image Transformer". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 4055–4064. URL: https://proceedings.mlr.press/v80/parmar18a.html.

[45] Rohit Prabhavalkar et al. "Automatic gain control and multi-style training for robust small-footprint keyword spotting with deep neural networks". In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2015, pp. 4704–4708.

[46] Archiki Prasad, Preethi Jyothi, and Rajbabu Velmurugan. "An Investigation of End-to-End Models for Robust Speech Recognition". In: *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2021, pp. 6893–6897. DOI: 10.1109/ICASSP39728.2021.9414027.

[47] Oleg Rybakov et al. "Streaming Keyword Spotting on Mobile Devices". In: *Interspeech 2020*. ISCA, 2020. DOI: 10.21437/interspeech.2020-1003. URL: https://doi.org/10.21437\%2Finterspeech.2020-1003.

[48] Leslie N. Smith and Nicholay Topin. "Super-convergence: very fast training of neural networks using large learning rates". In: *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*. Ed. by Tien Pham. Vol. 11006. International Society for Optics and Photonics. SPIE, 2019, p. 1100612. DOI: 10.1117/12.2520589. URL: https://doi.org/10.1117/12.2520589.

[49] David Snyder, Guoguo Chen, and Daniel Povey. *MUSAN: A Music, Speech, and Noise Corpus*. 2015. arXiv: 1510.08484 [cs.SD].

[50] Ming Sun et al. "Max-pooling loss training of long short-term memory networks for small-footprint keyword spotting". In: *2016 IEEE Spoken Language Technology Workshop (SLT)*. 2016, pp. 474–480. DOI: 10.1109/SLT.2016.7846306.

[51] Yu Sun et al. "Test-Time Training with Self-Supervision for Generalization under Distribution Shifts". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 9229–9248. URL: https://proceedings.mlr.press/v119/sun20b.html.

[52] A.P. Varga and R.K. Moore. "Hidden Markov model decomposition of speech and noise". In: *International Conference on Acoustics, Speech, and Signal Processing*. 1990, 845–848 vol.2. DOI: 10.1109/ICASSP.1990.115970.

[53] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[54] Xiong Wang et al. "Adversarial Examples for Improving End-to-end Attention-based Small-footprint Keyword Spotting". In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019, pp. 6366–6370. DOI: 10.1109/ICASSP.2019.8683479.

[55] Pete Warden. *Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition*. 2018. arXiv: 1804.03209 [cs.CL].

[56] Pete Warden and Software Engineer Google Brain Team Google. *Launching the speech commands dataset*. 2017. URL: https://ai.googleblog.com/2017/08/launching-speech-commands-dataset.html.

[57] Jin Xu et al. "Reluplex made more practical: Leaky ReLU". In: *2020 IEEE Symposium on Computers and Communications (ISCC)*. 2020, pp. 1–7. DOI: 10.1109/ISCC50000.2020.9219587.

[58] Cheng Yu et al. "Waveform-based voice activity detection exploiting fully convolutional networks with multi-branched encoders". In: *arXiv preprint arXiv:2006.11139* (2020).

[59] Mengjun Zeng and Nanfeng Xiao. "Effective Combination of DenseNet and BiLSTM for Keyword Spotting". In: *IEEE Access* 7 (2019), pp. 10767–10775. DOI: 10.1109/ACCESS.2019.2891838.

[60] Qiu-Shi Zhu et al. "A Noise-Robust Self-Supervised Pre-Training Model Based Speech Representation Learning for Automatic Speech Recognition". In: *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2022, pp. 3174–3178. DOI: 10.1109/ICASSP43922.2022.9747379.

[61] Qiu-Shi Zhu et al. "Robust Data2VEC: Noise-Robust Speech Representation Learning for ASR by Combining Regression and Improved Contrastive Learning". In: *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2023, pp. 1–5. DOI: 10.1109/ICASSP49357.2023.10095373.

Appendix A

Appendix A name

A.1 Babble Noise Results

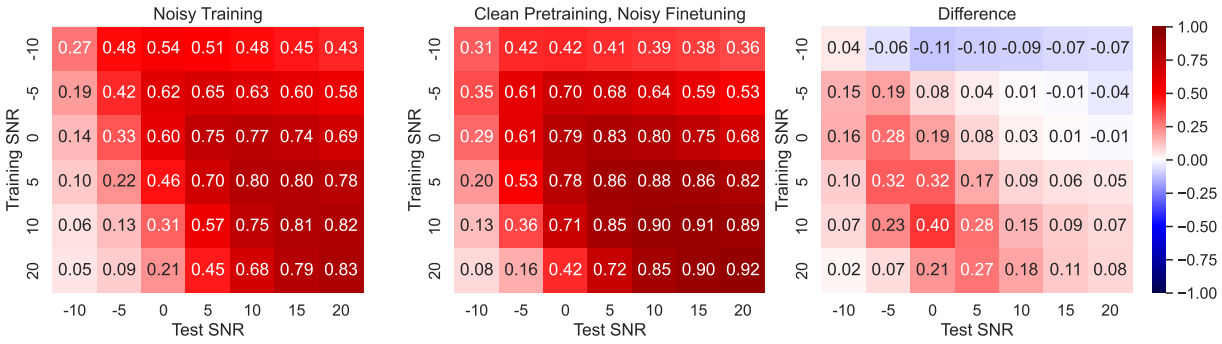

Figure A.1: KWT-3 results for BBL-noisy training and test. The left heatmap shows the results after noisy training, the middle heatmap shows the results after clean pertaining and noisy finetuning, and the right one shows the increment that clean pertaining offers.

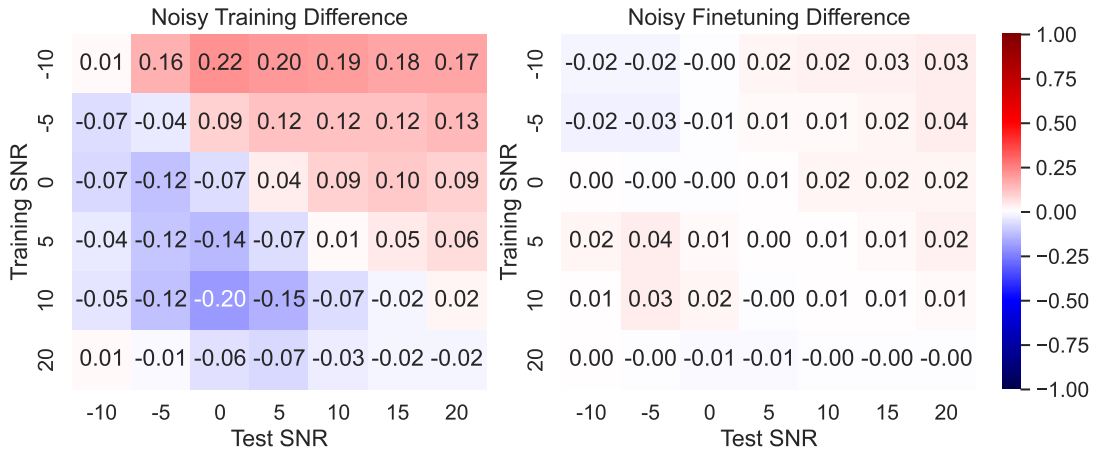

Figure A.2: The difference between KWT-3 and KWT-2 performance on BBL noisy data. The left heatmap shows the difference between KWT-3 and KWT-2 trained on BBL noisy data, and the right heatmap shows the difference between KWT-3 and KWT-2 pretrained on noisy data and finetuned on BBL-noisy data.

Figure A.3: KWT-3 results for BBL noisy and test. The left heatmap shows the accuracy of KWT-3 trained on noisy data. The middle heatmap shows the accuracy of KWT-3 trained on clean data, and the right one shows their difference.

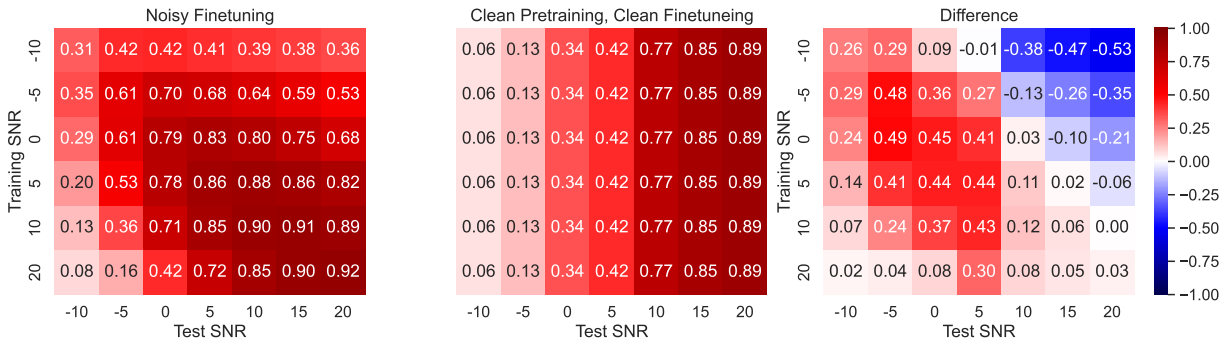

Figure A.4: KWT-3 results for BBL-noisy and test. The left heatmap shows the accuracy of KWT-3 pretrained on clean data and finetuned on noisy data. The middle heatmap shows the accuracy of KWT-3 pretrained and finetuned on clean data, and the right one shows their difference.