Karlsruhe Institute of Technology

# Detection and Classification of Sound Events in Automatic Speech Recognition

Master's Thesis
of


## Stefan Meintrup

KIT Department of Informatics
Institute for Anthropomatics and Robotics (IAR)
Interactive Systems Lab (ISL)


| | |
|---|---|
| Referees: | Prof. Dr. Alexander Waibel |
| | Prof. Dr.-Ing. Tamim Asfour |
| Advisors: | M.Sc. Carlos Mullov |
| | M.Sc. Christian Huber |

March 15, 2022 - September 1, 2022

# Abstract

Systems that generate transcripts from speech are called automatic speech recognition systems. The transcripts that are generated by these systems do usually not contain sound events such as laughing or clapping. However, when using generated transcripts as subtitles, it might be desirable to include specific sound events inside the transcript. In this work, we develop a deep learning automatic speech recognition system to generate transcripts from conversational speech that contain sound events.

We train an encoder-decoder, attention based, long short-term memory model to generate transcripts that contain breath, clearing throat, cough, laugh, lip smack, noise, and sniff sound events. Our experiments show that duplicating rare sound events in the training data significantly improves their detection and classification accuracy. This so-called upsampling improves the classification accuracy of breath, clearing throat, cough, lip smack and sniff sound events. For example, upsampling improves the classification accuracy of lip smack sounds from 45% to 82%. Additionally, we propose an efficient method to add sound event labels to popular datasets. These new labels improved the classification of cough sounds even further from 75% to 87%. However, the new labels also decrease the classification accuracy of sniff and lip smack sound events by 8% and 7% respectively. The best performing model detects and classifies clearing throat, laugh and lip smack sounds with over 80% and breath, cough, and noise sound events with over 70% accuracy.

# Zusammenfassung

Von Spracherkennungssystemen generierte Transkripte enthalten für gewöhnlich nur die gesprochenen Worte und keine Geräusche wie Klatschen oder Lachen. Wenn man solche Systeme zum Generieren von Untertiteln für Filme oder Videos benutzt, dann kann es jedoch gewünscht sein auch manche Geräusche zu transkribieren. In dieser Arbeit entwickeln wir ein Deep Learning Spracherkennungssystem um Transkripte von natürlicher Sprache zu erstellen, welche zusätzlich zu den gesprochenen Wörtern auch Geräusche enthalten.

Wir trainieren ein auf attention basierendes, encoder-decoder, bidirektionales, long short-term memory Modell, dessen Transkript Räuspern, Husten, Lachen, Schmatzen, Schniefen und Atemgeräusche enthält. Um die Geräuschklassifikation zu verbessern, duplizieren wir seltene Geräusche in den Trainingsdaten und verbessern dadurch die Klassifikationsrate von Räuspern, Husten, Schmatzen, Schniefen und Atemgeräuschen. Zum Beispiel wird dadurch die Klassifikationrate von Schmatzgeräuschen von 45% auf 82% gesteigert. Darüber hinaus entwickeln wir eine Methode um effizient Geräuschklassen zu bestehenden Datensätzen hinzuzufügen. Die hierdurch neu gewonnen Daten verbessern die Klassifikation von Hustengeräuschen um weitere 12% (von 75% auf 87%). Allerdings verschlechtern die neu hinzugefügten Daten wiederum die Geräusche Schmatzen und Schniefen um jeweils 7% und 8%. Das beste System erkennt und klassifiziert Räuspern, Lachen und Schmatzen zu über 80% und erkennt Husten und Atemgeräusche zu über 70%.

# Acknowledgements

First, I would like to thank Prof. Dr. Alexander Waibel for the opportunity to write my thesis at the Interactive Systems Lab. Not only did he get me excited about the topic, but he also made the manual labeling of the data possible. Furthermore, I would like to thank my advisors Christian Huber and Carlos Mullov for providing the pretrained model, the suggestion of upsampling the training data and for advising me during the whole thesis.

# Contents

# 1 Introduction

This chapter motivates the generation of transcripts that contain sound events, states the goal of this thesis and gives an overview of our contribution. Lastly, an outline of future chapters is given.

## 1.1 Motivation

Deep neural networks (DNNs) are widely used in the field of automatic speech recognition (ASR). Recently, sequence to sequence (s2s) models have started to achieve human performance in some automatic speech recognition benchmarks [20]. To achieve such performance, the DNNs are trained on hundreds of hours of speech data. Popular datasets include audio data and the corresponding transcript, but usually omit sound events or merge them into an unknown or noise class. Sound events can be sounds such as laughter, coughing or clapping. The lack of such sound event labels in conversational speech datasets makes the classification of sound events in conversational speech difficult. Still, the detection and classification of such sound events is interesting because the detection of a yawn or laughter can indicate the emotional state of a human. When humans yawn they might be tired or bored of a conversation. A laughter might indicate that the speaker is having a good time. Another application for transcripts that contain such sound events is the generation of subtitles in movies, videos or audio recordings. When deaf people do not see the speaker, they can not know whether or not the speaker is laughing unless the subtitles do include a laugh sound event.

## 1.2 The goal of this thesis

The goal of this thesis is to develop an ASR system that not only transcribes human speech, but also transcribes sound events such as laughter, breathing or coughing. A generated transcript might look something like this:

i totally agree with you <cough> excuse me.

The generated transcript does not include any punctuation or casing.

## 1.3 Contribution

We train an encoder-decoder, sequence to sequence, attention based, bidirectional, long short-term memory model to detect and classify breath, clearing throat, cough, laugh, lip smack, noise and sniff sound events during automatic speech recognition. Our first

attempt classifies clearing throat, laugh and noise sound events with over 78% accuracy. However, the model performs significantly worse on sniff, lip smack and cough sound events which do not occur frequently in the training data. To increase the model's performance on infrequent sounds, we duplicate utterances that contain rare sounds and reevaluate the model's performance. This upsampling of the training data increases the classification accuracy of cough, lip smack and sniff sounds by 25%, 38% and 15% respectively.
Since there are not many conversational speech datasets that contain a variety of sound events, we also try to efficiently add sound event labels to existing datasets to improve the model's performance even further. First, we use the best performing model to generate a transcript of the Switchboard and Fisher conversational speech datasets. Then, we use the attention scores of the model to extract transcribed sound events. Finally, we manually label these extracted sounds and merge them into the original dataset's transcript. This new data improves the classification of cough sounds by another 12%. However, the new labels also decrease the classification accuracy of sniff and lip smack sound events by 7% and 8% respectively. The best performing model detects and classifies clearing throat, laugh and lip smack sounds with over 80% and breath, cough, and noise sound events with over 70% accuracy.

## 1.4 Outline

In chapter two, concepts of automatic speech recognition and neural networks are introduced. Chapter three gives an overview of related work published in sound event and social signal detection. In chapter four, we describe the architecture of our ASR system and the dataset that is used to train the model's parameters. This chapter also includes a description of the data preprocessing and the experimental setup. Chapter five contains the evaluation of the training and the model's performance. In chapter six, a review and an outlook of future work is given.

# 2 Background

This chapter introduces automatic speech recognition and different neural network (NN) architectures.

## 2.1 Automatic speech recognition

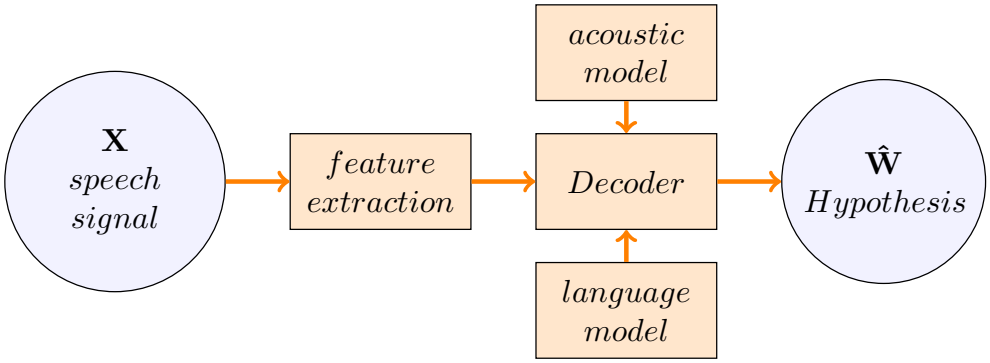


Figure 1: An ASR system based on [1]. Blue circles are input/output and the orange rectangles are part of the ASR system.

ASR systems interpret human speech. Figure 1 shows an ASR system's main components. In general, an ASR system's objective, given an observation $X = X_1X_2...X_N$, is to compute the word sequence $\hat{W} = w_1w_2...w_m$ that has the maximum posterior probability $P(W|X)$ [10]:

$$\begin{aligned}\hat{W} &= \arg\max_W P(W|X) \\ &= \arg\max_W \frac{P(W)P(X|W)}{P(X)} \\ &= \arg\max_W P(W)P(X|W).\end{aligned}$$

Here, $P(X|W)$ is computed by the so-called acoustic model and $P(W)$ is computed by the so-called language model. Neural networks that are used for ASR often combine the acoustic model, language model and decoder instead of modelling them independent of each other.
The following is a brief introduction to an ASR system's components.

### 2.1.1 Feature extraction

A given audio recording that, for example, is supposed to be converted to written text, has to be preprocessed first. Here, preprocessing is equivalent to the extraction of features that are relevant for the desired conversion into text. However, before extracting relevant

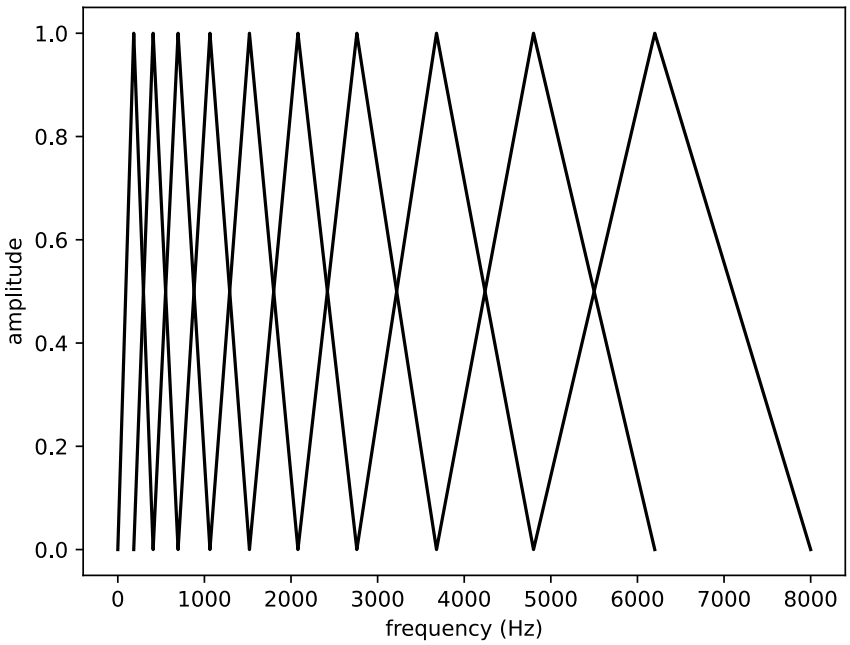


Figure 2: A 10-filter Mel-filterbank based on [13].

features it is important to make sure that the sample rate and the resolution of all audio files is the same.

A common preprocessing step is the conversion of the audio into a frequency representation by applying the Fourier transform resulting in a so-called power spectrum of the audio. It is also common to split the audio data into chunks of 20-30 ms length [11] that have an overlap. One could already use such a representation as input features for an ASR system, however, Mel Frequency Cepstral Coefficients (MFCC) [4] have proven to produce even better features.

The following explanation is based on [13]. MFCC emulate the human perception of frequencies, namely being very sensible at lower frequencies and differentiating less between higher frequencies. To compute the MFCC, the power spectrum is multiplied by a filterbank (see figure 2). After that, the energies in each filter are summed up. Since the human ear does not recognise loudness on a linear scale, the logarithm of the prior computed energies is taken. Lastly, it is desired to decorrelate the computed energies. The correlation between the energies stems from the overlap of the triangular filters in the filterbank. The decorrelation is done by taking the discrete cosine transformation of the energies. This leaves us with the MFCC of the audio signal as input for an ASR system (see figure 3 for an example).

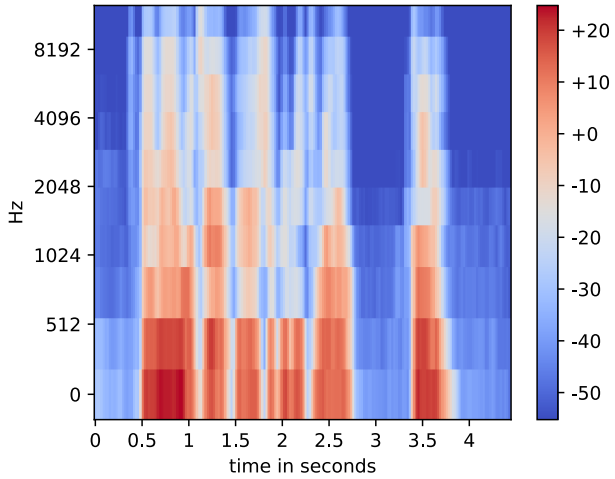


Figure 3: MFCC of an English sentence.

### 2.1.2 Language models

A neural ASR system is an ASR system that is using neural networks to decode the input features. Prior to this neural ASR systems, stochastic models have been widely used to decode the input features. The following briefly introduces stochastic language models (based on [10]). Stochastic language models estimate the probability $P(W)$ for a given word sequence $W = w_1 w_2 ... w_n$. Good stochastic language models can help an ASR system to differentiate between good and bad hypotheses to select the final hypothesis accordingly.

A typical stochastic language model is the so-called n-gram model. Given a string $W$, an n-gram model estimates the probability $P(W)$ of that string occurring in human speech as follows:

$$\begin{aligned} P(W) &= P(w_1, w_2, ..., w_n) \\ &= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \cdots P(w_n|w_1, w_2, ...., w_{n-1}) \\ &= \prod_{i=1}^{n} P(w_i|w_1, w_2, .., w_{i-1}), \end{aligned}$$

where $P(w_1)$ is the probability that the string starts with $w_1$ and $P(w_j|w_1, w_2, ..., w_{j-1})$ is the probability that $w_j$ follows the given $j-1$ other words. The n, in n-gram, points out how many pasts words $P(W)$ takes into consideration. The *unigram*, $n = 1$, takes only one word into consideration while the *bigram*, $n = 2$, takes the last word and the word before that into consideration. Lastly, a *trigram* takes the last word and two words before that into consideration.

A bigram estimates the probability of $W$ ="asr is fun" as follows:

$$P(asr\ is\ fun) = P(asr| < s >)P(is|asr)P(fun|is)P(< /s > |fun),$$

where <s> is an extra word to pad the beginning of a sentence and </s> is an extra word to pad the end of a sentence.
The probabilities are obtained from a training corpus that consists of millions of sentences. For a trigram, one counts all word pairs $C(w_{i-2}, w_{i-1})$ and triplets $C(w_i - 2, w_i - 1, w_i)$ to compute:

$$P(w_i|w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i)}{C(w_{i-2}, w_{i-1})}.$$

### 2.1.3 Sentencepiece

In order for an ASR system to output words that it has never seen before, one can split the words in the training corpus into smaller pieces. The ASR system can then output pieces of words instead of whole words. This enables the ASR system to create new words from those pieces. Byte pair encoding [6], a string encoding scheme, is frequently used to create such pieces. Sentencepiece [16] is a language independent subword tokenizer and detokenizer that is based on BPE. Given a vocabulary size, sentencepiece computes a vocabulary of that size. Each word in the vocabulary is a common character sequence of the input string. This allows an ASR system to output words that it has never seen before.

### 2.1.4 Word error rate

The word error rate (WER) is a widely used performance metric for ASR systems and focuses on three errors [10]:

- Substitution: The correct word has been substituted with an incorrect word
- Deletion: The correct word is omitted by the ASR system
- Insertion: An additional word has been added by the ASR system

Once the number of substitutions, deletions and insertions have been computed, the WER is defined as [10]:

**Definition 2.1** (Word error rate)**.**

$$Word\,error\,rate = 100\% \cdot \frac{\#substitutions + \#deletions + \#insertions}{Number\,of\,words\,in\,the\,correct\,sentence},$$

where #deletions denotes the number of deletions in the hypothesis.

### 2.1.5 Perplexity

The WER of a model is not sufficient to estimate its performance on unseen data. Consider two models A and B that have the same WER on a specific test dataset $T = \{x_1, ..., x_M\}$. Model A might be a hundred percent certain about every prediction while Model B might be very uncertain about every prediction. Model B is expected to generalize worse on new data. This is why the perplexity, measuring the certainty of a network, is used to estimate when to stop training a network.

**Definition 2.2** (Perplexity [13])**.**

$$perplexity(T) = 2^{H(T)}$$

$$H(T) = -\frac{1}{M}\sum_{m=1}^{M} \log\left(p(x_m)\right)$$

$p(x_m)$ denotes the probability that the model assigns to the correct label.

## 2.2 Artificial neural networks

Artificial neural networks are a popular machine learning technique to classify data [31]. The data to be classified is fed into the neural network and the network computes a prediction of a class that corresponds to the input data. Here, the input data might be an audio recording and the computed class may be a descriptor of a sound (laughter for example). This section introduces different types of neural network architectures and their differences.

### 2.2.1 The Rosenblatt perceptron

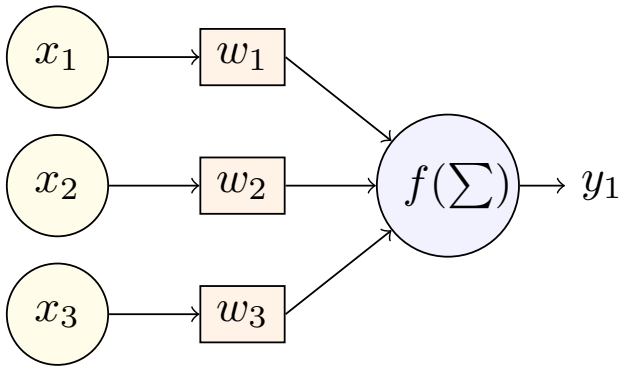


Figure 4: A Rosenblatt perceptron with input: $x \in \mathbb{R}^3$, weights $w \in \mathbb{R}^3$ and the output neurons value $y_1 \in \mathbb{R}$.

The Rosenblatt perceptron [23] (figure 4) is a small artificial neural network. It consists of only one output neuron. Not only does the output neuron sum up the products of the inputs $x_i$ and the corresponding weights $w_i$, it also feeds this sum into a function:

$$y_1 = f(x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3).$$

In the case of classification, the function $f$ is a threshold function that, for example, might classify all values greater than 0 to be of one class and all other values to be of another class. When used for classification, the Rosenblatt perceptron operates as a binary, linear classifier. However, a Rosenblatt perceptron can also be used as a regressor. In this case, the function $f$ may be the identity function.
To set up the weights of the Rosenblatt perceptron, or any artificial neural network, we initialize them to random values. The next section explains how these weights are trained using Backpropagation.

### 2.2.2 Gradient Descent and Backpropagation

The Backpropagation algorithm is used to train a neural network, meaning Backpropagation is used to iteratively change the neural network's weights to improve its output value when fed specific inputs. To do so, Backpropagation utilizes Gradient Descent to alter the weights of a neural network [24]. First, we need a function that estimates how large the error of the network, given a specific input, is. For example, we might want to train a Rosenblatt perceptron as in figure 4. A typical error function is the sum squared error [24]:

**Definition 2.3** (Sum squared error)**.** Given the outputs of a neural network $o_{mn}$, with $m \in \{1, 2, ..., M\}$ and $n \in \{1, 2, ..., N\}$, where $N$ is the number of output neurons of the neural network and $M$ is the training dataset size. Let $l_{mn}$ be the desired output of neuron $n$ given input $m$, then the sum squared error is defined as:

$$SSE = \sum_{m=1}^{M} (\sum_{n=1}^{N} (o_{mn} - l_{mn})^2).$$

To use Gradient Descent, we first compute the derivative of the SSE with respect to a specific weight, using the chain rule:

$$\frac{\partial SSE}{\partial w_j} = \frac{\partial SSE}{\partial o} \cdot \frac{\partial o}{\partial w_j},$$

then we update the weights:

$$w_j = w_j - \delta \frac{\partial SSE}{\partial w_j}.$$

$\delta$ is the so-called learning rate and is used to scale the update speed of the weights.
The Backpropagation algorithm helps to efficiently apply Gradient Descent updates to a neural network.

### 2.2.3 Optimizer

When the training dataset is very large, it takes a long time to compute the network's error on the whole training set. This means that during training, it could take a long time until the network's weights can be updated using Backpropagation. Since it can take many updates to train a network, stochastic Gradient Descent is widely used.

**Stochastic Gradient Descent** works by sampling a minibatch from the training dataset, computing the error of the network over that minibatch and then updating the weights using Backpropagation [8]. This leads to more frequent updates of the network's weights per iteration over the training dataset. One iteration over the training dataset is called an *epoch*.

Adaptive learning rate optimization algorithms use a separate learning rate for each parameter of the network [8]. During training, each parameter is automatically adapted. The specific optimization algorithm determines how the parameters are automatically adapted. **Adam**, for example, is an optimization algorithm that estimates the first-order moment of the gradient to update the learning parameters [8].

### 2.2.4 Multioutput perceptron

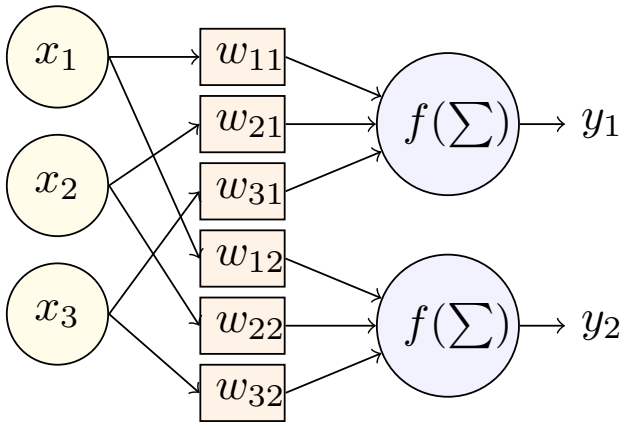


Figure 5: Multioutput perceptron with input: $x \in \mathbb{R}^3$, weights $w \in \mathbb{R}^{3\times 2}$ and the output neuron's values $y \in \mathbb{R}^2$.

There are multiple limitations to the Rosenblatt perceptron when it comes to solving classification tasks. Since a binary classifier can only predict two classes, a single Rosenblatt perceptron is insufficient to solve tasks with more than two classes. However, if one uses multiple Rosenblatt perceptrons, then one can solve classification tasks with an arbitrary amount of classes. If there are $N$ classes, then one can merge $N$ Rosenblatt perceptrons into one neural network (see figure 5 for an example with two classes). Now, one could

select the Rosenblatt perceptron with the highest value and output its corresponding class as the classification result. However, taking the maximum of the network prevents Gradient Descent since the max function is not differentiable. To compute a differentiable probability distribution over all the possible output classes, a softmax layer is added to the end of the NN:

**Definition 2.4** (Softmax layer)**.** A softmax layer takes the $N$ outputs of a neural network's output layer and computes:

$$softmax(o)_j = \frac{exp(o_j)}{\sum_{n=1}^{N} exp(o_n)}, \text{ with } j \in \{1, ..., N\}$$

as the new outputs of the network.

The softmax function ensures that all outputs are between 0 and 1 and add up to 1. If the desired output class $l \in \{1, ..., N\}$ of input $i$ is given, then we can compute the $SSE$ using the to $l$ corresponding one hot vector [8].

**Definition 2.5** (One-hot vector)**.** Let $h \in \mathbb{R}^N$ be a one-hot vector that corresponds to a task with $N$ classes and a label $l \in \{1, ..., N\}$. Then the one-hot vector $h$ is defined as the $l$-th unit vector.

Given the one-hot vector $h$ of label $l$, we can now compute the $SSE$ of the neural network with the softmax layer's values $s \in \mathbb{R}^N$:

$$SSE = \sum_{n=1}^{N} (s_n - h_n)^2.$$

Instead of the SSE, which is mostly used for regression tasks, the cross entropy loss is frequently used for classification tasks:

$$CEL = -\sum_{n=1}^{N} h_n \cdot log(s_n).$$

This allows us to train a perceptron with multiple outputs using Backpropagation.

Another limitation of a Rosenblatt perceptron is that it is a linear classifier and hence can not solve non-linear classification problems. In order to solve non-linear classification problems, we introduce multilayer perceptrons.

### 2.2.5 Multilayer perceptrons

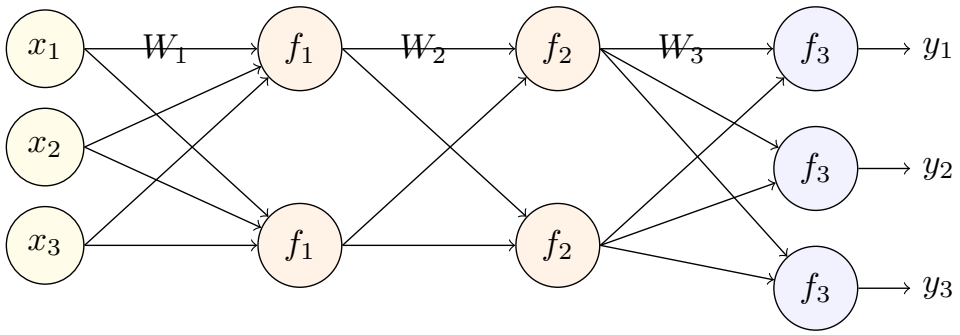


Figure 6: Multilayer perceptron with input: $x \in \mathbb{R}^3$, weight matrices $W_1 \in \mathbb{R}^{3\times2}$, $W_2 \in \mathbb{R}^{2\times2}$ and $W_3 \in \mathbb{R}^{2\times3}$ and the output neurons' values $y \in \mathbb{R}^3$.

The limitations of the Rosenblatt perceptron led to the development of the multilayer perceptron [17] (figure 6). A multilayer perceptron that consists of three layers can classify arbitrary decision surfaces. However, in order for the multilayer perceptron to decide non-linear decision surfaces, the functions, computed in the neurons, have to be non-linear. These functions are called activation functions. A classic example of an activation function is the so-called logistic sigmoid function.

**Definition 2.6** (Logistic sigmoid)**.**

$$\sigma(x) = \frac{1}{1 + exp(-x)}, \text{ with } x \in \mathbb{R}.$$

Another activation function is the Swish activation function [21]. The Swish activation function includes the sigmoid function.

**Definition 2.7** (Swish)**.**

$$f(x) = x \cdot \sigma(\beta x), \text{ with } x, \beta \in \mathbb{R}.$$

Even though three-layer multilayer perceptrons can, in theory, compute any decision surface, adding even more layers has been proven to be useful in practice.
One drawback of multilayer perceptrons is that they are not shift-invariant under translation in time and frequency [29]. For ASR systems, the input data might be an audio recording of conversational speech. The recording might start a few seconds before the subject started vocalizing the first words of the sentence. If the training has been done with examples where the recorded subject always started vocalizing the first word in the first second of the recording, then the network may be unable to transcribe the speech of slightly delayed sentences. Time-delay neural networks (TDNNs) [29], widely known as convolutional neural networks (CNNs), introduce shift-invariance in time and frequency dimension.

### 2.2.6 Convolutional neural networks (Time-delay neural networks)

TDNNs [29], also known as CNNs, are introducing shift-invariance in time and frequency dimension. Let Figure 7 be an input to a neural network. A trained network may have found that all training data has a pattern in the upper left corner that allows the network to predict the training label. If the layer is not a convolutional one, then the network will only recognise such patterns in the upper left corner. However, if a person has a significantly lower voice, then the same pattern could occur in the lower left corner (see figure 8). Additionally, if the speaker pauses for a few seconds (see figure 9), then the pattern may occur further to the right. A 2d-convolutional layer is shift-invariant in both time and frequency, meaning that a convolutional layer could find the pattern in all three cases. In figure 10, each rectangle represents the application of a so-called convolutional filter.

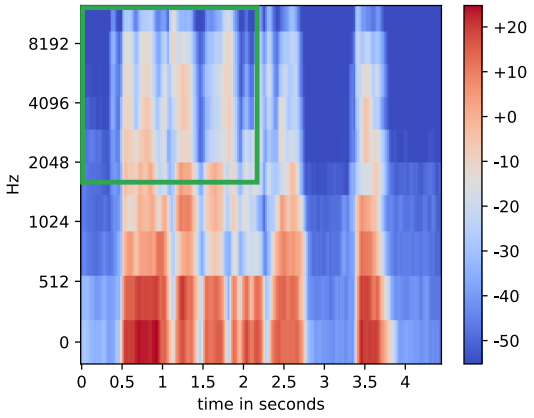


Figure 7: MFCC input features where a NN searches for a pattern in the rectangle.

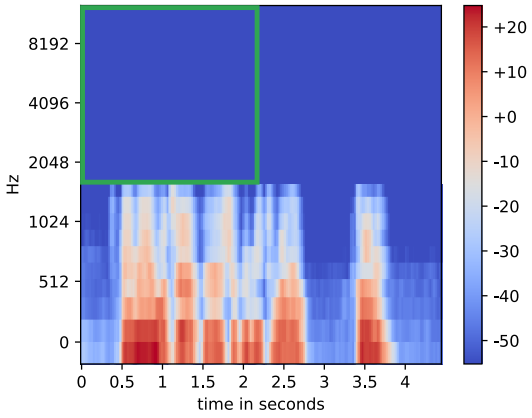


Figure 8: MFCC input features with lower frequency input.

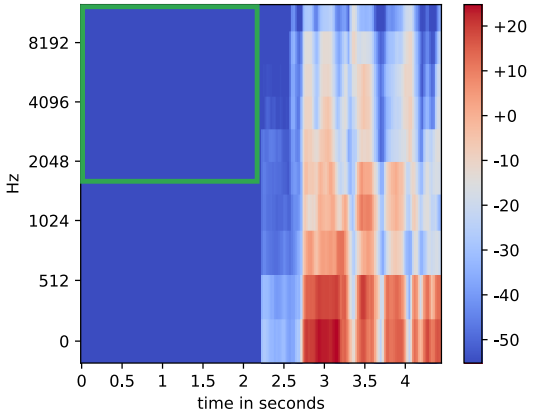


Figure 9: MFCC input features where the input is delayed.

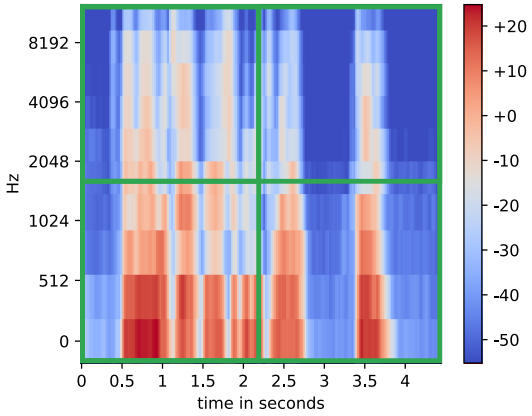


Figure 10: MFCC input on which a convolutional filter is applied 4 times.

Applying the filter four times leads to four neurons in the next layer. The rectangle size

is determined by the filter size and the shift between two applications of the same filter is determined by the so-called stride. In figure 10, the stride is equal to the height of the rectangle in y-direction and equal to the width of the rectangle in x-direction. If a filter is shifted to the right (according to the stride of the convolutional layer) it could exceed the input width. That's why the input is usually padded with zeros to ensure a perfect fit of the filter.
Although convolutional neural networks are an improvement to regular multilayer perceptrons, they are hardly used to map sequences to sequences [27]. Recurrent neural networks (RNNs) are widely used to map sequences to sequences.

### 2.2.7 Recurrent neural networks and sequence to sequence systems

Recurrent neural networks [25] can be applied to long sequences of variable length, while that would be impractical for networks like the multilayer perceptrons or CNNs [8]. Recurrent neural networks allow for connections that form a loop (see figure 11).

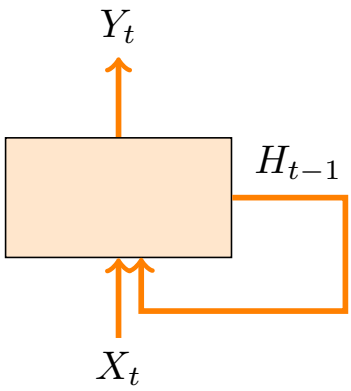


Figure 11: A recurrent neural network. $X_t$ is the input at timestep t, $Y_t$ is the output at timestep t and $H_{t-1}$ consists of the value of hidden neurons at timestep $t-1$.

It is not obvious how to use Gradient Descent in a NN that contains a loop. However, when we use recurrent neural networks in a s2s model, as in figure 12, we do not see loops anymore.

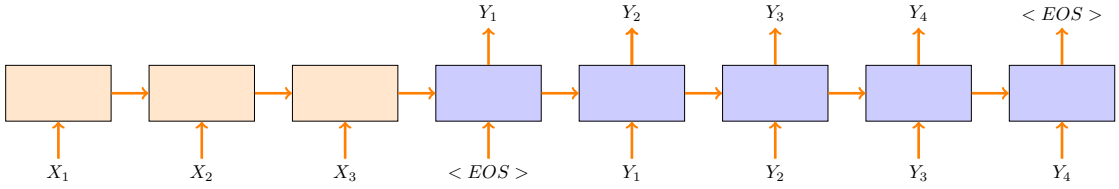


Figure 12: s2s model based on [27]. Orange parts form the encoder and the blue parts form the decoder. <EOS> is a special token and stands for end of sequence.

The encoder's RNN has been rolled out over time, meaning that each rectangle is a copy of the encoder's RNN. Each copy propagates its hidden state to the next timestep's RNN copy instead of having a self loop. Once a sequence is finished, the encoder's RNN is no longer rolled out. Since each rectangle is a copy of the same RNN, the weights in all RNNs are the same. However, when computing a gradient, the derivation could differ between the RNN copies. To avoid this, all RNN weights are considered to be shared. When updating the weights, the same update is applied to every weight, assuring that they will keep the same value. For example, this could be done by averaging all the computed updates of a shared weight. This procedure is called Backpropagation through time. When training a network on a long sequence, the computed gradients for the weights early in the model are harder to learn because they can get close to zero and hence hardly ever change. Long short-term memory neural networks (LSTMs) are used to increase the encoder's capability of learning long-distance dependencies.

### 2.2.8 Long short-term memory neural networks

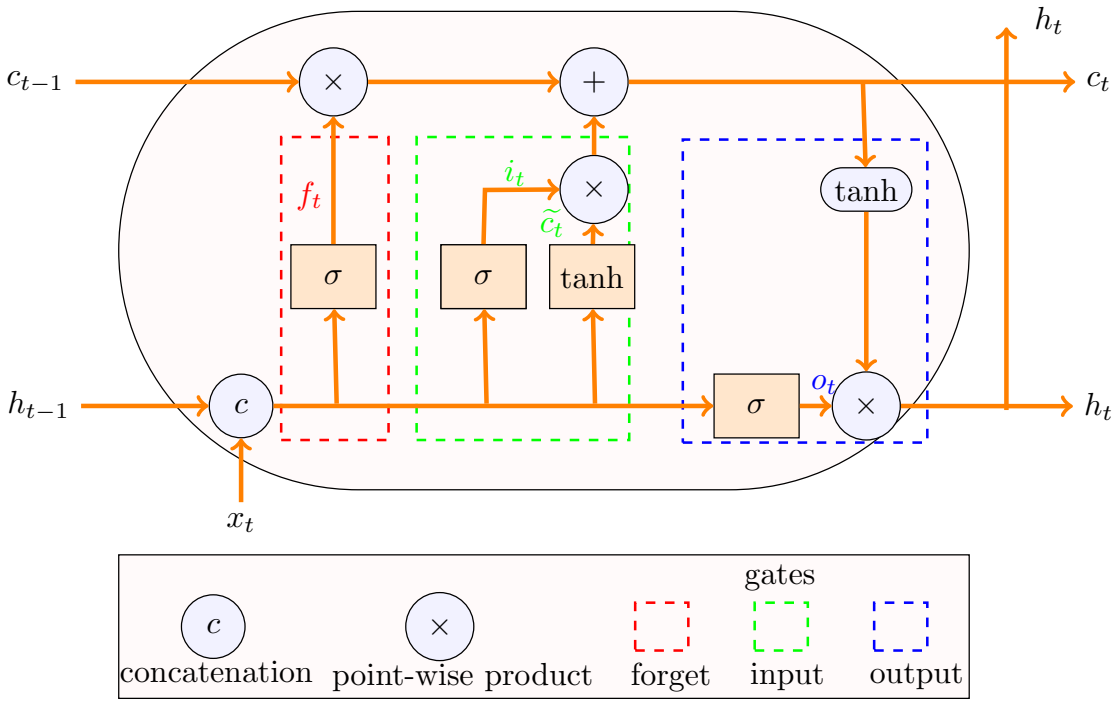


Figure 13: An LSTM cell based on [32]. $c_{t-1}$ and $c_t$ are the cell states. $h_{t-1}$ and $h_t$ are the outputs of the LSTM. $x_t$ is the input to the cell.

LSTMs [9] are used in s2s networks that are supposed to learn long-distance dependencies. LSTMs prevent the gradient from becoming very close to zero (vanishing) or becoming very large (exploding). An exploding gradient leads to large changes in the weights of a NN during training. This can lead to large jumps on the decision surface and can inhibit

efficient training. On the other hand, vanishing gradients lead to very little change in the weights of a NN. This can lead to insignificant jumps on the decision surface and hence to a very long training duration.
The connections through time in an LSTM are gated, changing the multiplied factors in each time step [8]. This helps to prevent the gradient from vanishing or exploding. The cell state of the LSTM cell is also called a memory cell. It is this cell state that allows the LSTM to maintain information over a long sequence. Overall, an LSTM cell does include three gates (see figure 13). The $\sigma$ in figure 13 denotes a sigmoid activation function that also includes a bias addition and matrix multiplication, leading to the following computations [32]:

$$
\begin{aligned}
f_t &= \sigma(U_f x_t + W_f h_{t-1} + b_f) \\
i_t &= \sigma(U_i x_t + W_i h_{t-1} + b_i) \\
\widetilde{c}_t &= \tanh(U_{\widetilde{c}} x_t + W_{\widetilde{c}} h_{t-1} + b_{\widetilde{c}}) \\
c_t &= f_t \times c_{t-1} + i_t \times \widetilde{c}_t \\
o_t &= \sigma(U_o x_t + W_o h_{t-1} + b_o) \\
h_t &= o_t \times \tanh c_t,
\end{aligned}
$$

where $\times$ denotes the element-wise product.

### 2.2.9 Bidirectional and multilayer LSTMs

The information in an unidirectional LSTM can only flow from left to right. This means that when cell $t$ computes $h_t$, it can not use any information about inputs of later timesteps. When an LSTM is used to translate a sentence then this limitation is unnecessary, if the whole sentence is given right away. Here, the knowledge of words that occur later in the sentence might lead to better hidden state representations. This is where bidirectional LSTMs can help. Bidirectional LSTMs compute $h_t$ twice and independently, once from left to right and once from right to left. Both $h_t$ are later combined and used as the hidden state representation of timestep $t$.
Multilayer LSTMs consist of multiple LSTM cells stacked on top of each other. The hidden state of a prior layer is fed into the next layer and the last layer forms the output of the LSTM.

### 2.2.10 Attention

The decoder of a sequence to sequence model computes a fixed size representation of the input sequence. Bahdanau et al. (2014) [2] suggest that this fixed size representation is a bottleneck of the encoder-decoder approach to neural machine translation. To overcome

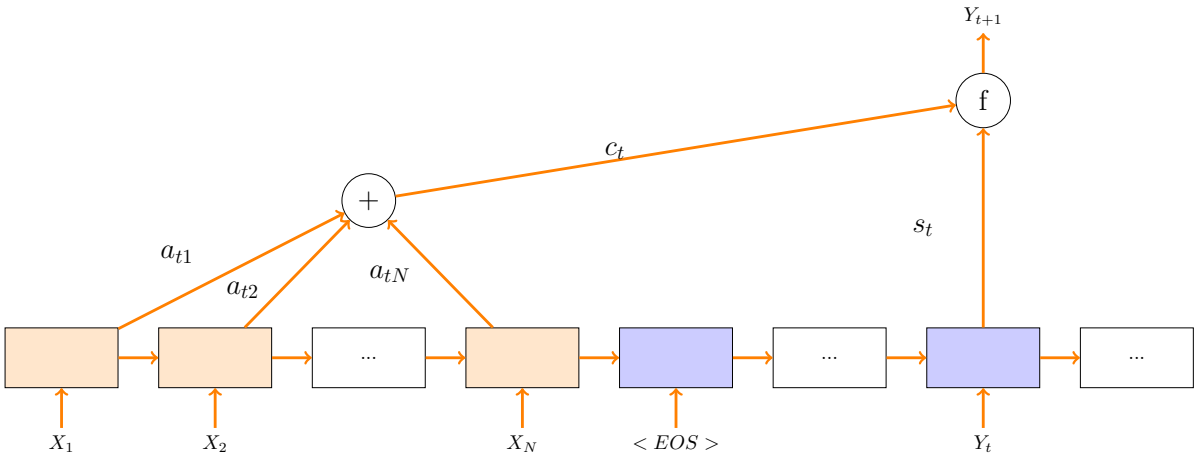


Figure 14: Attention at time step t+1 based on [2].

this bottleneck the authors suggest the use of an attention layer (see figure 14). Here, each encoder hidden representation $h_n$ influences $Y_{t+1}$ explicitly through the context vector $c_t$:

$$c_t = \sum_{n=1}^{N} \alpha_{tn} h_n.$$

The weight $\alpha_{tn}$ is defined as

$$\alpha_{tn} = \frac{exp(e_{tn})}{\sum_{k=1}^{N} exp(e_{tk})},$$

with

$$e_{tn} = a(s_t, h_n).$$

$a$ is a so-called alignment model which can be computed by its own neural network. Computing the weighted sum over the encoder hidden states is also called soft attention. Hard attention would be the selection of the hidden state that has the highest attention score.

### 2.2.11 Beam-search

At each decoder timestep in a s2s system the probability of each class in the output vocabulary is computed. Computing all possible hypotheses would lead to an exponential growth in computation compared to always picking the most probable class. Using beam-search, the B most likely hypotheses are kept for the next decoding step. Once the $< EOS >$ token is appended, the sequence is no longer part of the partial hypotheses. Instead, such a sequence is now part of the completed hypotheses [27]. Once all hypotheses are finished, one can pick the most likely one as a final prediction.

### 2.2.12 Generalization, regularization and dropout

A model that performs well on previously unseen inputs is said to generalize well [8]. Any modification to the network that is intended to improve generalization, instead of the training error, is called a regularization technique [8]. Dropout is a regularization technique that, during training, randomly removes non-output units from a model [8]. If, for example, the dropout is set to 20% and dropout is applied to the input units, the dropout is a random noise that is applied to the training data and (on average) deletes 20% of it. To drop out a unit in a multilayer perceptron, it is multiplied by zero.

### 2.2.13 Dropconnect

Dropout removes neural network units by multiplying them with zero. Dropconnect removes neural network weights by setting them to zero [30].

### 2.2.14 Early stopping

In theory, the training of a network is only finished when the error of a network is zero or at least very close to it. However, this might lead to bad generalization of the model. When a model performs very well on the training dataset, but performs poorly on a test dataset, the model is said to have overfit to the training data. To avoid this overfitting, a third dataset is introduced, the so-called validation dataset. When the error on the validation dataset stops improving, the training is stopped to avoid overfitting. This is called early stopping.

### 2.2.15 Finetuning and freezing layers

If a model has been successfully trained for a specific task like the recognition of animals in a picture, one could try to use the same model architecture for the recognition of cats in a picture. However, training the model might take a very long time if the model is trained from scratch. To reduce training time, one could use the already trained animal recognition model, change the output classes from animal to cat and start training with the weights of the already trained model. Taking another model as a starting point for training is called finetuning.
During finetuning, the weights that are used as a starting point are changed. When it is not desired to change the weights of one layer, then the weights of that layer can be frozen. A frozen layer's weights are not changed during training and hence always remain the same. This is done when one thinks that the model that is used as a starting point already computes useful features and hence should not be changed.

# 3 Related Work

When it comes to the detection and classification of sound events that occur in human speech, there has been research that solely focuses on sound events and research that combines automatic speech recognition and social signal detection.

## 3.1 Sound event detection without ASR

Sound event detection has been studied for single events like breathing or laughter. In 2018 the authors of [5] stated that there has been almost no research on breath event detection for spontaneous speech. Instead of focusing on spontaneous speech, previous studies focused on read speech or songs. In [5] the authors focus on the detection of inhaled breath events in Japanese telephone conversations. They specifically analyse the waveforms and spectrograms of breath events and propose a breath event detection algorithm that is based on a Guassian Mixture Model (GMM) and a Support Vector Machine (SVM).

Compared to breath events, there has been more research conducted on the detection of laughter events in spontaneous speech. The dataset that is used in this thesis is the ICSI Meeting Corpus [19] and it has been used in several studies that focus on detecting laughter events. Here, Support Vector Machines (SVMs) [14], Guassian Mixture Models (GMMs) [28], Hidden Markov Models (HMMs) [3] and neural networks [15] have been used. Studies on other datasets also use multimodal models to detect laughter [22][26]. Multimodal meaning the use of both an audio and a video stream as input to the sound event detection system.

When it comes to the detection and classification of many different sound events, the audioset dataset is one of the largest datasets [7]. The dataset consists of over 2 million Youtube videos which are labeled with 527 labels. There are labels like music, car, bird, speech, but also breathing, sniff and laughter. However, google research provides a quality estimate for each sound class and sounds like sniff or throat clearing only have an estimated quality of 38 and 33 percent respectively. This quality estimation is based on random samples that have been manually labeled by humans. The data mostly consists of the labels music and speech which both have a quality estimation of 100 percent. Googles yamnet is a CNN based classifier that is trained on the audioset dataset and labels 960 millisecond long audio chunks of a given wavefile as one of 521 classes.

## 3.2 Social signal detection combined with ASR

In [12] an end-to-end BILSTM approach for social signal detection and automatic speech recognition has been proposed. Here, social signals are split into four events: laughter, fillers, backchannels and disfluencies. Fillers are vocalizations like "ah", "eh", "uhm" and are used by the speaker when he needs time to think. Backchannels are vocalizations like "yeah", "right", "sure" and "okay". They are used by the speaker to communicate to the listener that the speaker pays attention and, in some cases, agrees with what they were saying. Disfluencies describe repetitions, corrections and false starts. In the proposed system, fillers, disfluencies, backchannels and even laughter are describing the transcribed speech[12]:

$$D_{start}\ Tha\ D_{end}\ F_{start}\ uh\ you\ know\ F_{end}\ L_{start}\ that's\ like\ ...\ L_{end}.$$

$D_{start}$ and $D_{end}$ denote the start and end of a disfluency ($F_{start}$ corresponds to filler and $L_{start}$ corresponds to laughter). Notably, the additional detection of laughters, fillers, backchannels and disfluencies did not impact the performance of the ASR significantly. Meaning that the social signals neither helped nor inhibited the transcription of vocalized words.

This thesis combines sound detection/classification with ASR, but without focusing on social signals. Instead, the classified sounds laughter, breath, cough, sniff, throat clearing and lip smack are treated like new words in the transcript and do not describe transcribed words.

# 4 Approach

This chapter describes the network architecture of the model and the data that is used in the experiments.

## 4.1 Data

We used the International Computer Science Institute Meeting Corpus [19] as the dataset for our experiments. The data consists of 70 hours of 75 recorded meetings between ICSI members in 2001. The meeting rooms were equipped with distant and close-talking microphones. Each audio channel and its transcript are given. Not only does the data include transcribed text, but also transcribed noises such as different kinds of laughters, breathing, microphone noise or paper noise.

### 4.1.1 Detailed information about the data

Meetings have been recorded in different environments. One meeting room was set up at Columbia University, one at SRI International and one at the University of Washington. Each participant had one separate microphone. In addition to that, six far-field microphones were set up. Also, the connection type of each microphone is given, namely whether or not it is wired or wireless. The data is available, either merged into one single channel or separated into up to 15 different channels. Each file has a sampling rate of 16 kHz. The length of each utterance is determined by "the first longer audible pause" in a channel's recording. The data is available in XML format and includes identifiers for the speakers and microphones. Furthermore, the individual speakers have short descriptions including sex, education, ethnicity and their native language.

## 4.2 Data preprocessing

### 4.2.1 Audio data preprocessing

The audio data has been converted into wavfiles with a sampling rate of 16 kHz. The resulting files consist of a single channel. The MFCCs of the wavfiles are used as the input features of the model.

### 4.2.2 Transcript data preprocessing

The whole transcript has been transformed into a lower case transcript. Words that end with "-" correspond to words that have not been finished by the speaker. This happens, for

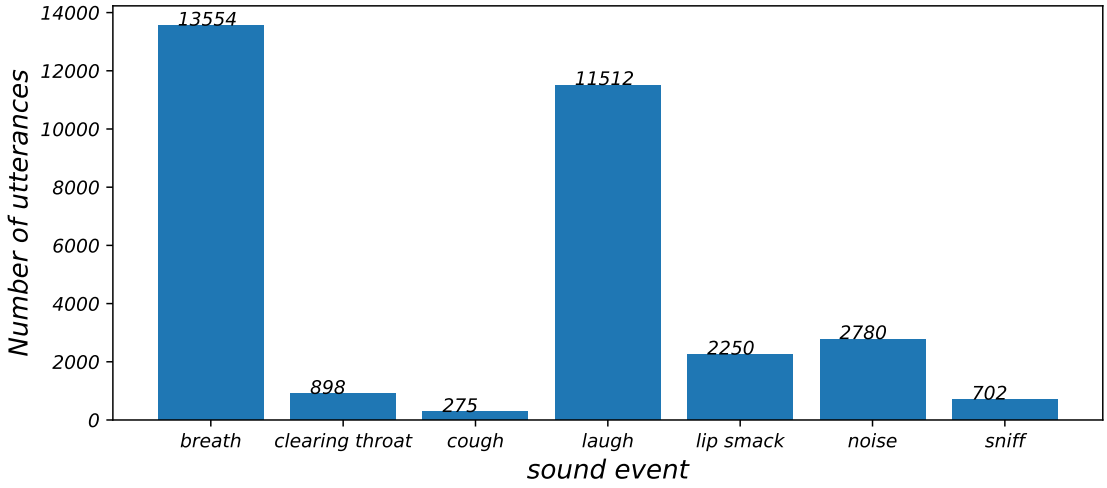


Figure 15: Number of utterances in the original data that contain a specific sound event.

example, when a speaker gets interrupted or is thinking while speaking. Those unfinished words are deleted from the transcript. Since the original transcript's sound event labels are a textual description, there exist a lot of different descriptions. Descriptions that contain "emphasis", "uncertain", "pronounce", "pause" or "foreign" have been ignored. All sounds that occurred less than 50 times have been merged into the noise sound class. In the end, all microphone noises, drinking, gasping, sneezing, door, clicking, sighing, hiccups, whistles and click noises have been included in the noise class. Figure 15 shows the sound events that are left in the transcript after preprocessing.

### 4.2.3 Data split

We created two datasets. One dataset contains all utterances and one dataset only contains utterances that consist of one word or one sound.

**Sentence-level dataset:** Two hours of the data are used as test data and one hour is used as validation data. The two datasets contain two and three complete meetings respectively and are selected to have a similar sound event distribution as the original data. Figure 16 shows the sounds that are contained in the training set. Figure 17 shows the sounds that are contained in the two validation meetings. Figure 18 shows the sounds that are contained in the three test meetings.

**Word-level dataset:** This dataset only consists of utterances that contain either one word or one sound. Each word has been replaced with a <word> class. Figures 19-21 show the training, validation and test data distribution. Most sounds occur over 50% in isolation.

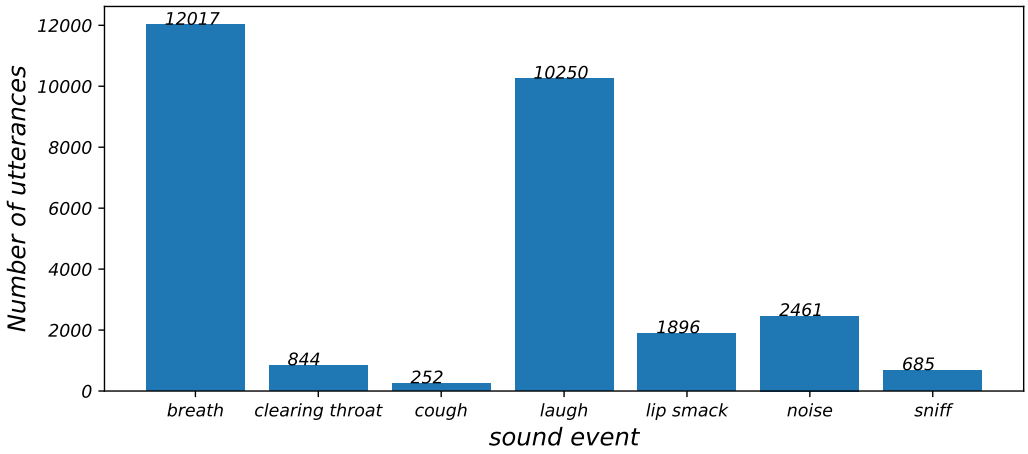


Figure 16: Number of utterances in the sentence-level training data that contain a specific sound event.

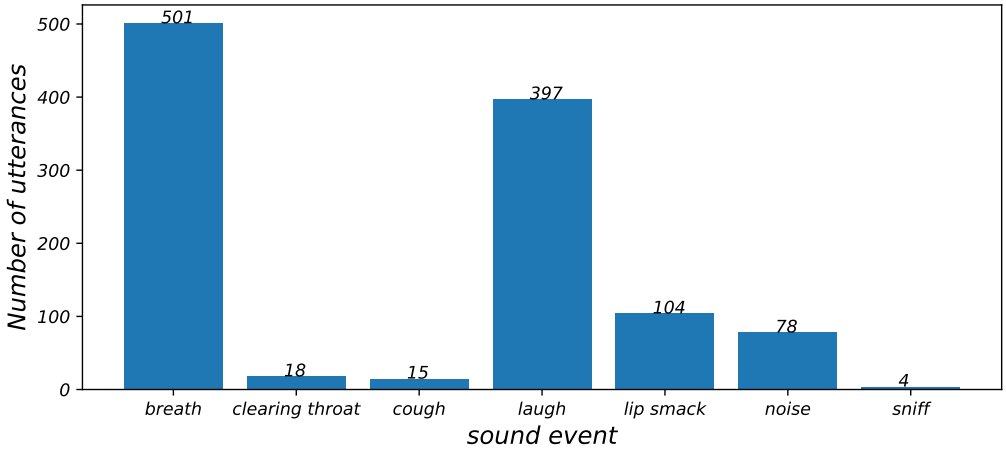


Figure 17: Number of utterances in the sentence-level validation data that contain a specific sound event (Meetings Bmr024, Bmr006).

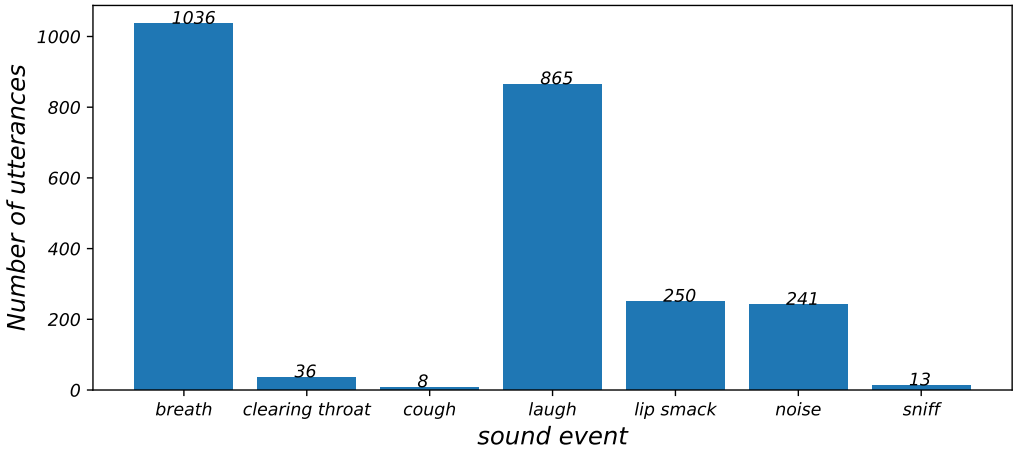


Figure 18: Number of utterances in the sentence-level test data that contain a specific sound event (Meetings Bmr031, Bmr023, Bmr028).

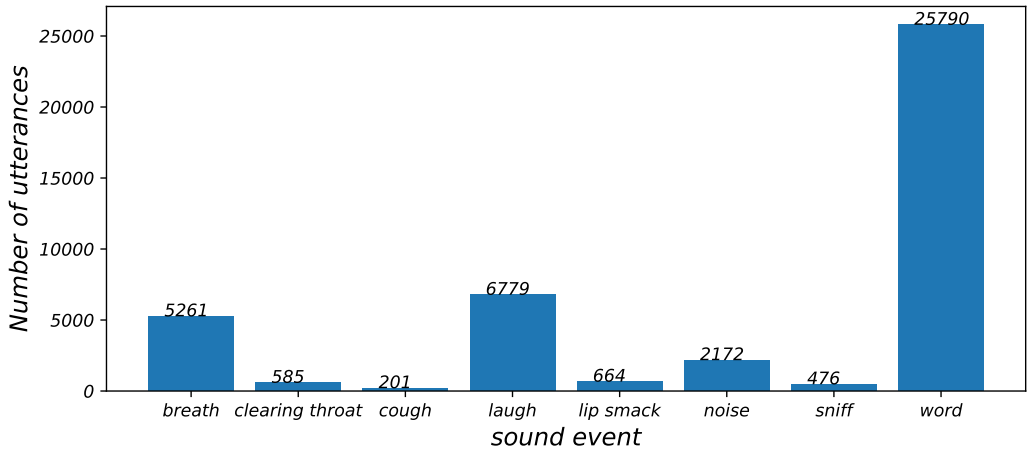


Figure 19: Number of utterances in the word-level training data that contain a specific sound event.

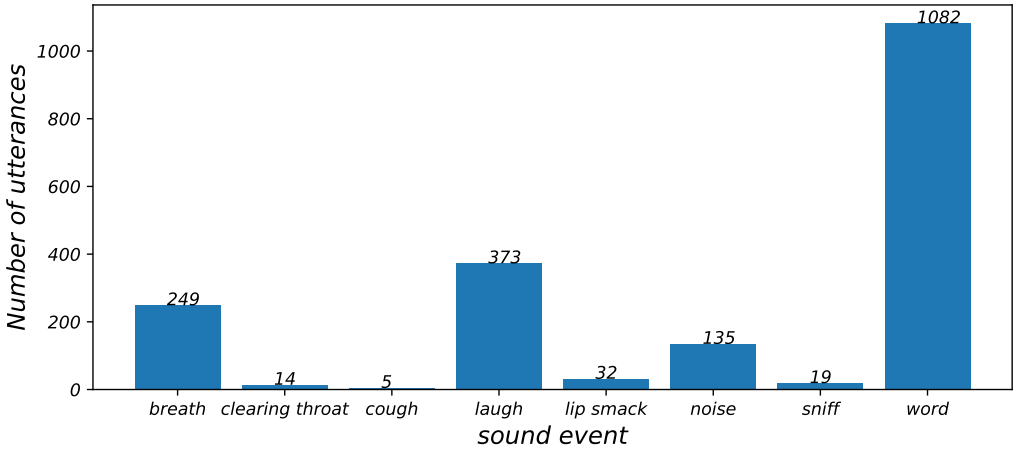


Figure 20: Number of utterances in the word-level validation data that contain a specific sound event (Meetings Bmr031, Bmr022, Bro005).

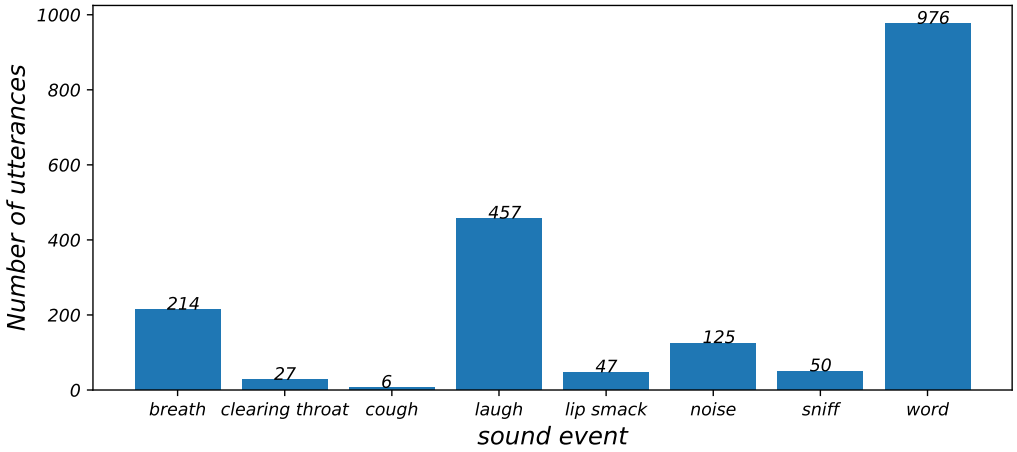


Figure 21: Number of utterances in the word-level test data that contain a specific sound event (Meetings Bmr027, Bed014, Bmr028).

## 4.3 Model

The experiments have been conducted using a sequence to sequence encoder-decoder neural network (see figure 22). The encoder consists of a bidirectional multilayer LSTM. The decoder consists of a unidirectional multilayer LSTM with an attention layer. Note that the encoder and decoder are not directly connected. Instead, the decoding vector is used as the query vector of the attention over the encoder hidden states.
Before the MFCC parameters are fed into the encoder LSTM, two 2d-CNN (TDNN) layers are applied. These two layers detect frequency and time-invariant patterns in the MFCC input parameters of the encoder.

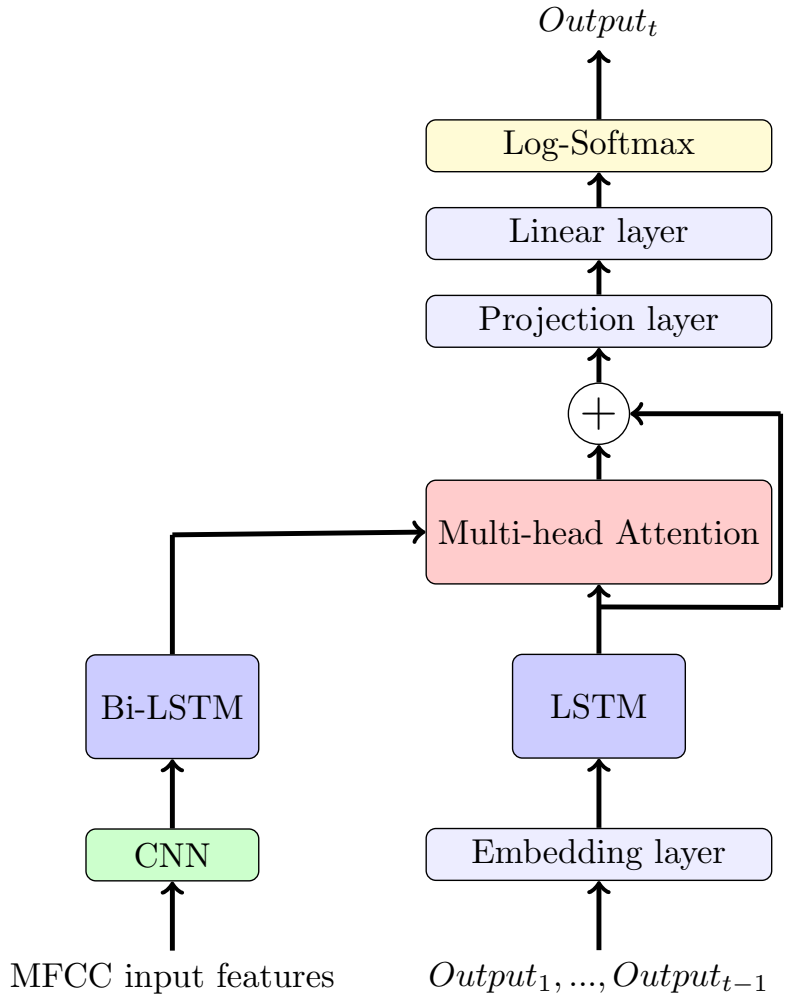


Figure 22: Computation of the t-th output of the s2s model.

### 4.3.1 BPE

Instead of training the model to predict whole words, the sentencepiece tokenizer was used to create a byte pair encoding of the transcript. The sound classes have been added after the byte pair encoding. Sound classes are directly predicted by the network.

## 4.4 Experimental setup

In the following, a description of the model's hyperparameters is given.

### 4.4.1 Model parameters

**Encoder parameters:** The MFCC vectors of dimension 40 get fed into a TDNN layer. The encoder contains two consecutive TDNN layers. Both layers have a kernel size of 3 in the time and frequency dimension. The stride of both layers is 2 in the time and frequency dimension. Both layers use the Swish activation function. The first layer consists of 1 input channel and 32 output channels. The second layer consists of 32 input and 32 output channels. The encoder BILSTM consists of 6 layers, has a hidden vector dimension of 1024 and has its dropout and dropconnect set to 0.3.
**Decoder parameters:** The embedding vector dimension is 512 and the embedding is scaled by $\sqrt{512}$. The embedding layer includes a dropout layer with the dropout set to 0.15. The LSTM has two layers and a hidden vector dimension of 1024 with its a dropout and dropconnect set to 0.2. The attention layer consists of one attention head. The attention layer also includes a dropout layer with the dropout set to 0.2. The projection layer is a fully-connected layer and projects the vector of size 1024 to a vector of size 256. The linear layer is a fully connected layer with an output size of 4000 BPE vocabulary tokens plus the sound class tokens.

# 5 Evaluation

The evaluation of our approach consists of three parts: First, we evaluate the model using the sentence-level training data and an upsampled version of the sentence-level training data. Then, we train and evaluate the model on the word-level dataset where the model only has to differentiate between the sound event class and a word class. Finally, the model is evaluated using new data which is generated by a prior iteration of the model and then labeled by two persons.

## 5.1 Training the model on the sentence-level training data

We used the sentence-level training data to train the BILSTM in four different ways. We trained the model from scratch and finetuned a baseline model that has been trained on the datasets: Common Voice, Europarl, Fisher, How2, Hub4, Libri, MustC, Switchboard, Tedlium and Voxforge (roughly 4000 hours of data). This baseline model has an WER of 5.5% on the Tedlium test dataset. We also tried finetuning the model with a frozen encoder or with a frozen decoder. The embedding and the linear layer of the decoder are not frozen.

Since we combine ASR and sound detection/classification, we evaluate the model's performance on both tasks. To evaluate the ASR accuracy, the WER of each model is shown in table 1. Transcribed sounds are ignored during the WER computation. The sound event detection/classification accuracy of the four models is shown in figures 25, 45, 46 and 47. We added the nothing class for utterances where a sound occurs either only in the hypothesis or only in the transcript. The training accuracy is shown in figure 23 and includes the transcribed sounds. Figure 24 shows the training perplexity of the four models.

| Model | WER |
|---|---|
| from scratch | 26.6% |
| baseline model | 28.1% |
| finetuning | 16.3% |
| finetuning freeze decoder | 16.8% |
| finetuning freeze encoder | 21.7% |

Table 1: Word Error Rate of models that are trained on the sentence-level training data and the WER of the baseline model.

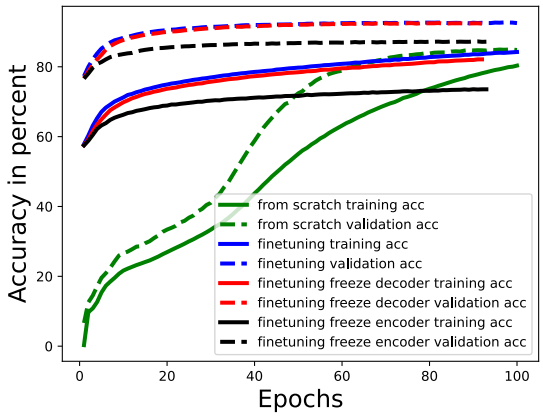


Figure 23: Accuracy of models that are trained on the sentence-level data.

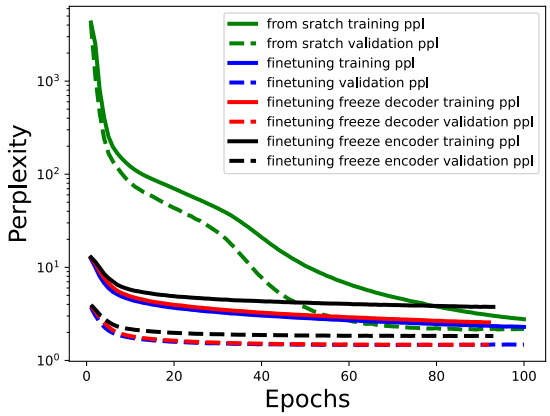


Figure 24: Perplexity of models that are trained on the sentence-level data.

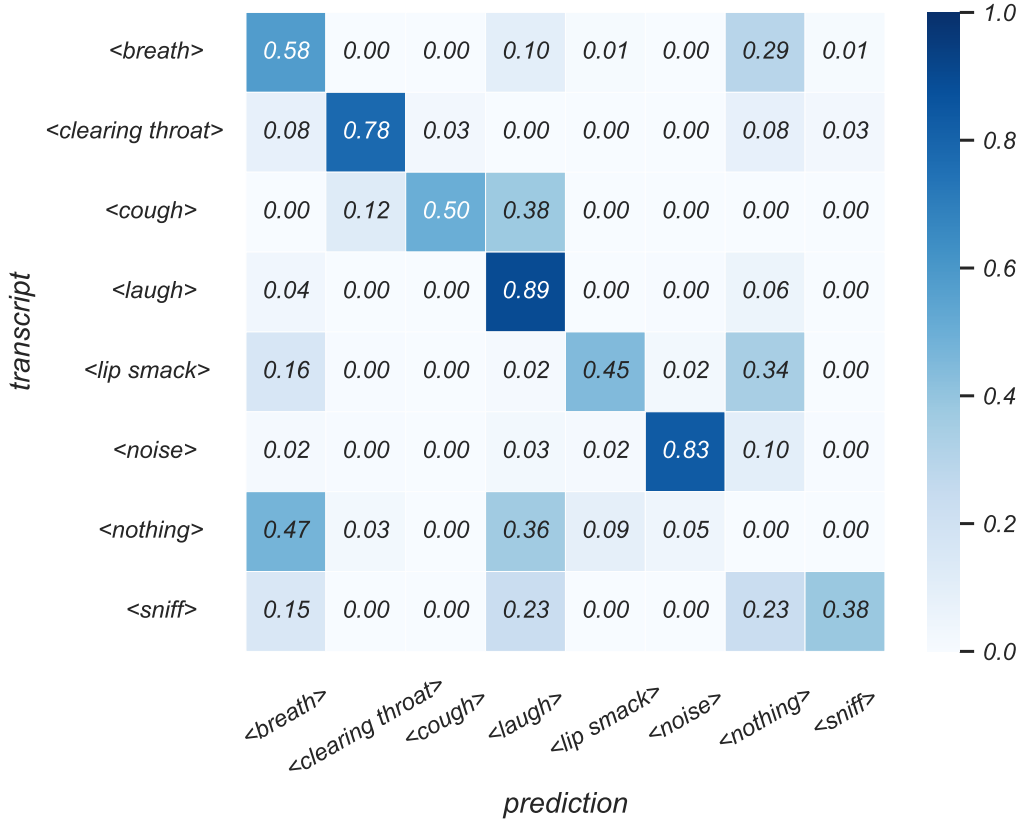


Figure 25: Sound classification accuracy of the finetuning model which is trained on the sentence-level training data.

The finetuning model is the model with the lowest WER and classifies the sounds with a similar accuracy as the model that has been trained from scratch and the finetuning model with the frozen decoder. The model with the frozen encoder did miss a lot more of the annotated sounds compared to the other models (see figures 25, 45, 46 and 47). Laugh, noise and clearing throat sound events are classified with the highest accuracy.

Breath, sniff and lip smack sound events are often not labeled at all (29%, 34% and 23% respectively). Also, many breath and laugh sounds events have been predicted by the models even though there was no such sound in the transcript (47% and 36% respectively).

## 5.2 Upsampling the training data

Since the finetuning model is the best-performing model, we choose it as the model that we train for the next experiments. Figure 25 shows that sniff, lip smack and cough sound events are classified with an accuracy smaller or equal to 50%. These sound events occur significantly less often than frequent ones like laugh or breath. To test if this under representation of some sound events is the reason for their poor classification accuracy, the following experiment has been conducted. Sentences that contain a throat clearing, cough, lip smack or sniff sound are multiplied so that each sound roughly occurs 10000 times. This leads to all the sounds, except breath and noise, occurring similarly often (see figure 26). Noise and laugh are not upsampled because their classification accuracy is already over 80%. We call this upsampled training set: 'upsampled training data v1' to distinguish it from other upsampled training datasets. The WER of the model that is trained on this dataset does not change significantly (see table 2) since the sounds are ignored during the WER calculation. Figure 27 shows the change in the sound classification accuracy that is caused by upsampling (see figure 25 for the original and figure 48 for the upsampled accuracies).

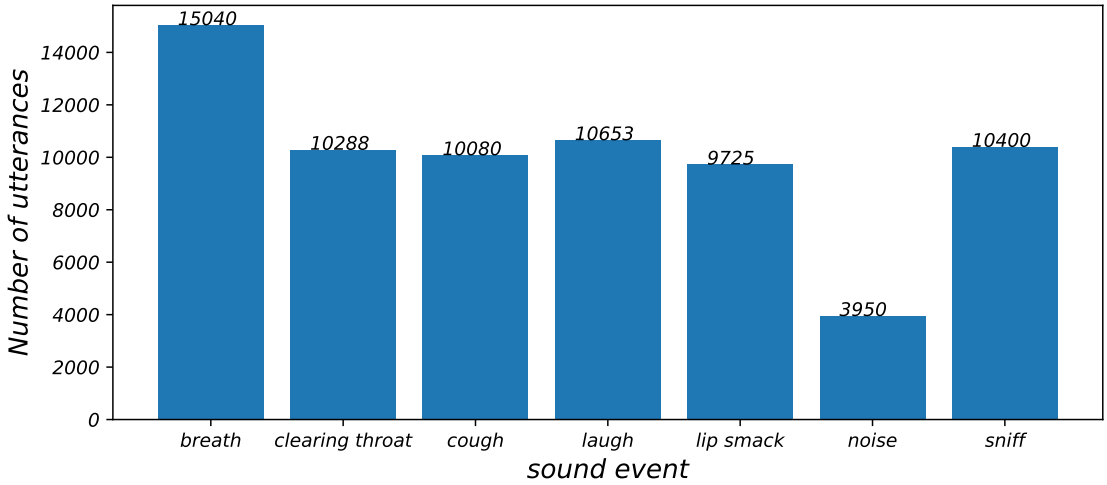


Figure 26: Number of utterances in the upsampled training data v1 dataset that contain a specific sound event.

| Model | WER |
|---|---|
| Finetuning upsampled training data v1 | 16.7% |

Table 2: Word Error Rate of the model trained on the upsampled training data v1 dataset.

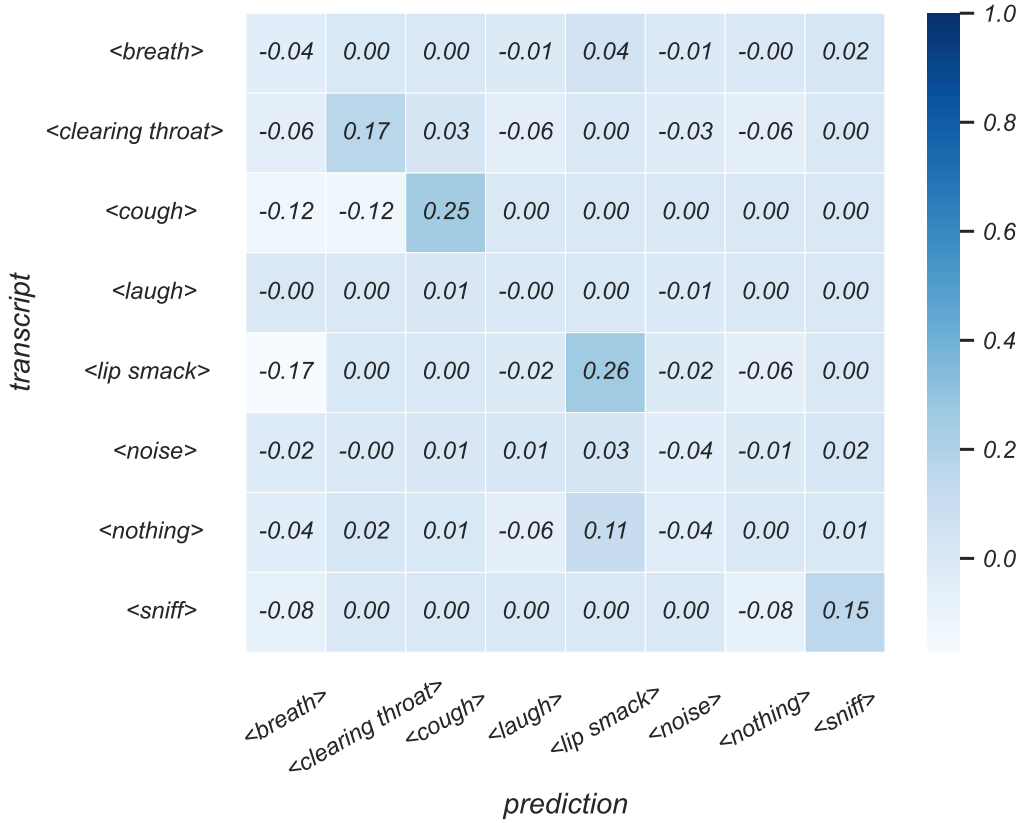


Figure 27: Improvement in sound event prediction accuracy by training the finetuning model on the upsampled training data v1 dataset.

The model that was trained on the upsampled training data v1 dataset performs significantly better on the sounds: Clearing throat, cough, lip smack and sniff.
After showing that upsampling the sounds increases the sound classification performance, we want to analyse if further upsampling the sounds always leads to even better performance. Specifically, we double the amount of upsampled sniffs since the sniff classification has the most room to improve. Figure 49 shows that an 8% improvement in the classification accuracy of sniff sounds is offset by a 2%, 3% and a 2% worse prediction of breath, clearing-throat and laughter sounds. Simply upsampling sniffs further does not seem to be sufficient to improve the model's overall classification accuracy.
After experimenting with different upsampled datasets, the best-performing model was trained on the training data distribution depicted in figure 28 and has the accuracy shown in figure 29 and table 3.

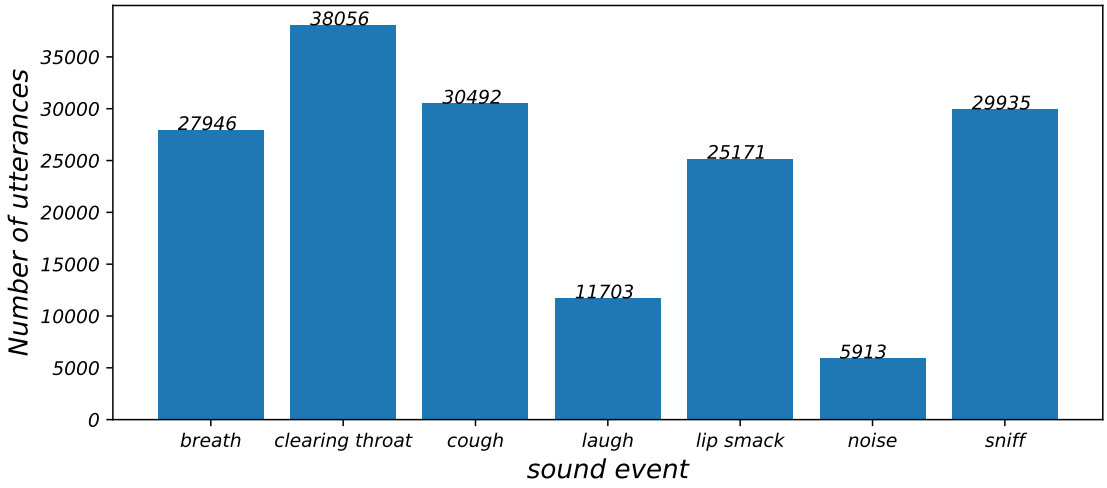


Figure 28: Number of utterances in the upsampled training data v2 dataset that contain a specific sound event.

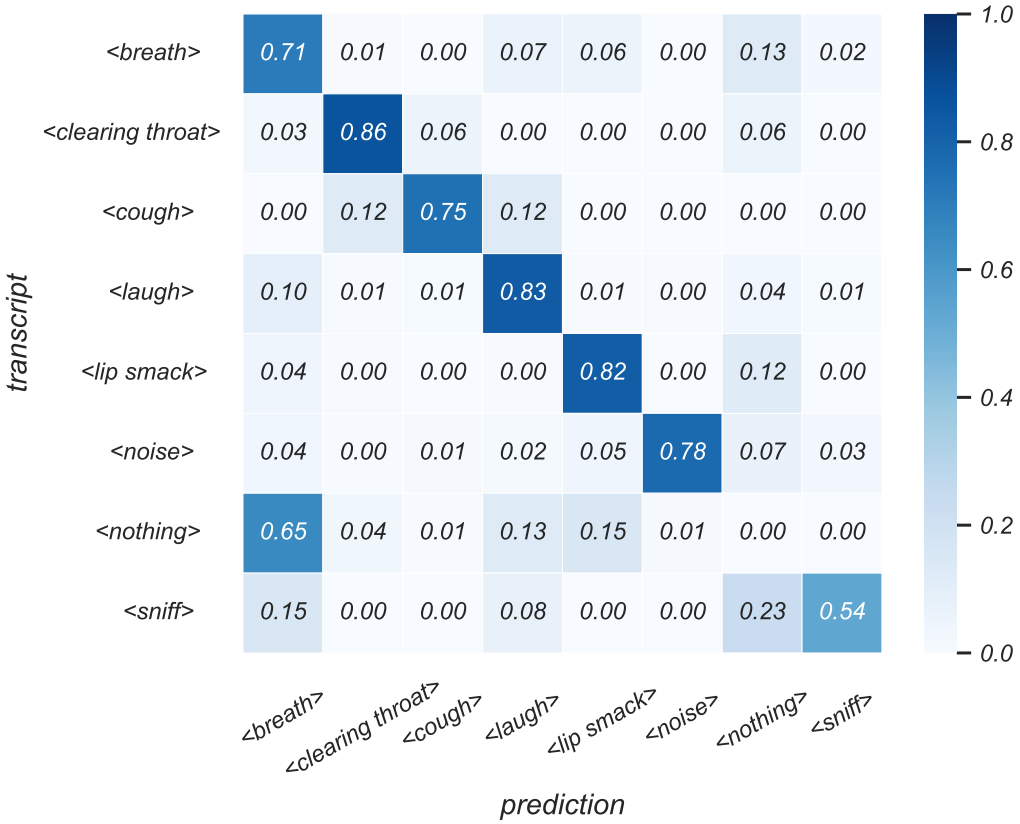


Figure 29: Sound classification accuracy of the finetuning model which is trained on the upsampled training data v2 dataset. This is the overall best performing model.

| Model | WER |
|---|---|
| Finetuning upsampled training data v2 | 16.8% |

Table 3: Word Error Rate of the finetuning model trained on the upsampled training data v2 dataset.

## 5.3 Word-level classifier

If we could segment an utterance's input features into the individual words and sounds then we could assign the sound label with a word-level classifier. We trained the same sequence to sequence model on the word-level dataset to classify individual words and sounds. We changed the output vocabulary from 4000 plus the sound labels to the sound labels plus one word class. Figure 30 shows the accuracy of the word-level classifier.

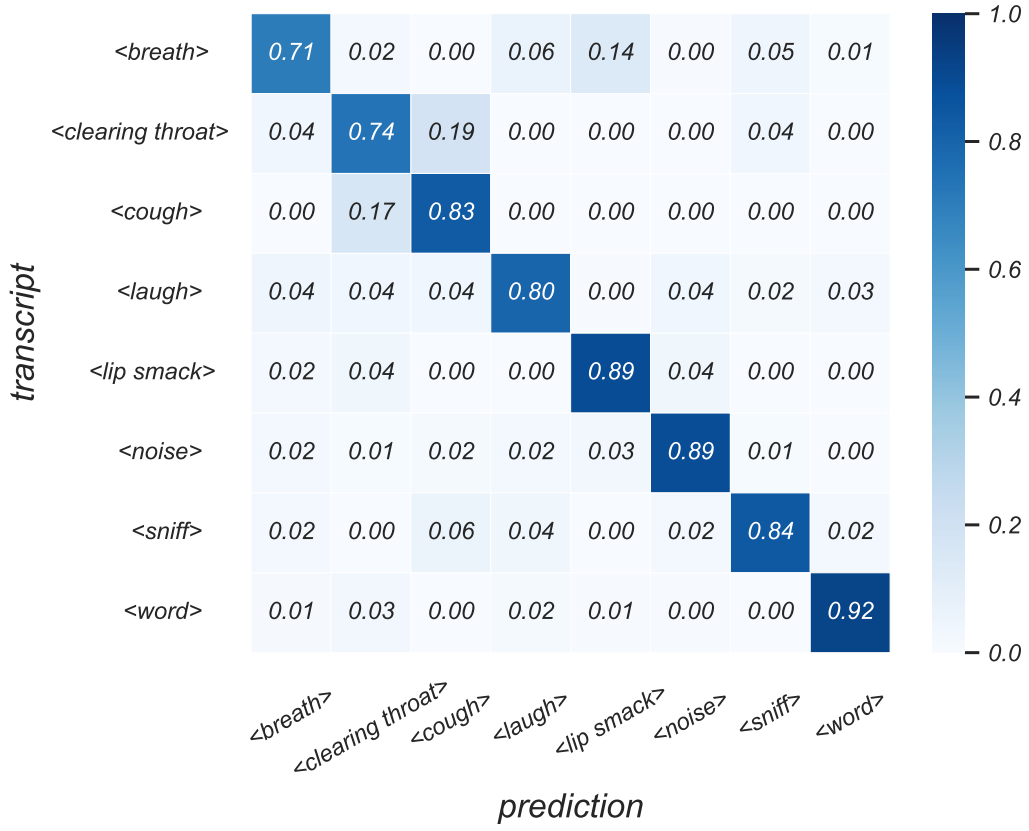


Figure 30: Sound classification accuracy of the finetuning model on the word-level data.

The model recognises words with a 92% accuracy. At a sentence-level, sniff sounds have been classified with an accuracy of 54% while the word-level model classifies sniffs with an accuracy of 84%.

We tried segmenting the input utterances into the individual sounds and words using a CNN. We output a word or sound class for every 50ms of input data. However, we were

not able to obtain a model that creates a high quality segmentation. Instead, our CNN model outputs are alternating between many different labels which does not lead to a good segmentation.

## 5.4 Data generation

The goal of the following experiments is to label more data by using the trained model to label sounds from datasets that do not have labels such as cough and sniff.
First, we tried to generate new training data from the Switchboard dataset to decrease the WER of the model. We used the best performing model so far to label all utterances in the Switchboard dataset. Then we took all utterances that had no sound in their hypothesis and added such utterances to the training dataset. However, finetuning the model on this dataset did not improve the WER of the model. This might be because the baseline model was trained on the Switchboard dataset.
To generate transcripts for new utterances that contain sound events, we could listen to labeled utterances and then correct them manually. However, sounds like sniff and lip smack are very short. Simply listening to a sound and assigning it to one of a few classes would be way faster than listening to whole sentences. This motivated the data generation strategy depicted in figure 31.

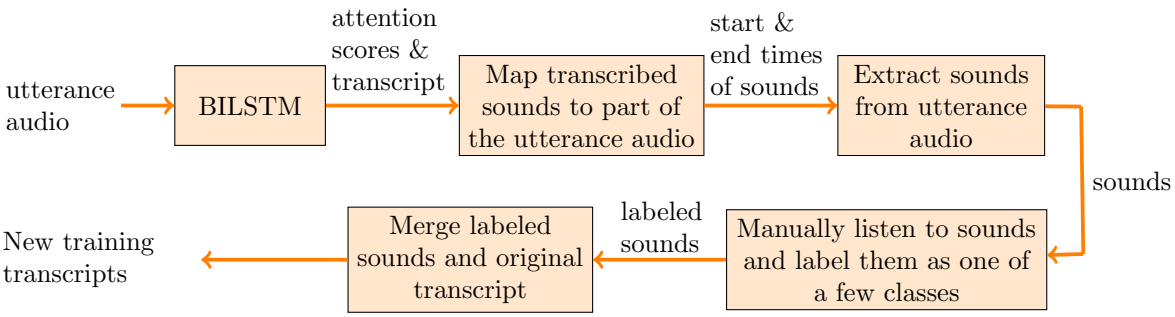


Figure 31: Labeling pipeline.

### 5.4.1 Mapping transcribed sounds to the utterance audio

In machine translation, the attention scores of a single attention layer have been used to find an alignment between the source and the target sentence [18] (see figure 32 for an example). The attention score for a particular word in the hypothesis is usually higher for the words that directly influence its translation. The example shows that the

hypothesis word "Ich" has the highest attention score for its direct translation into the source language "I". However, it is not guaranteed that the attention score always aligns the hypothesis words with their direct translation.

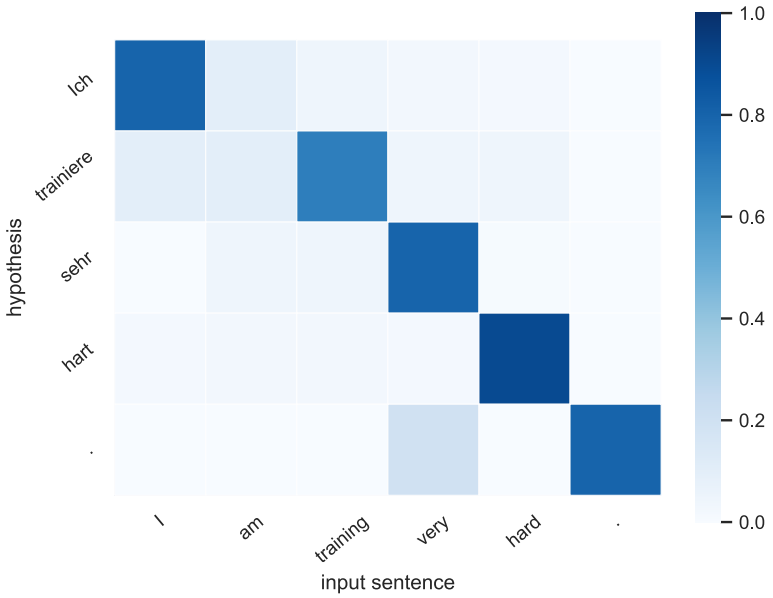


Figure 32: Attention score alignment in machine translation.

We use the attention scores to align the model-generated transcript and the utterance audio. First, we take a look at the attention score distribution of different sounds to determine whether or not it makes sense to implement similar extraction strategies for different sound types. Figures 33-38 suggest that the attention score distribution for different sounds is similar. Hence, we try a unified sound extraction strategy to extract transcribed sounds.

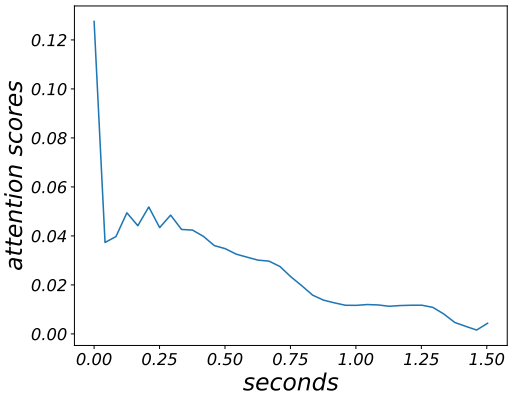


Figure 33: Attention score distribution of a cough sound.

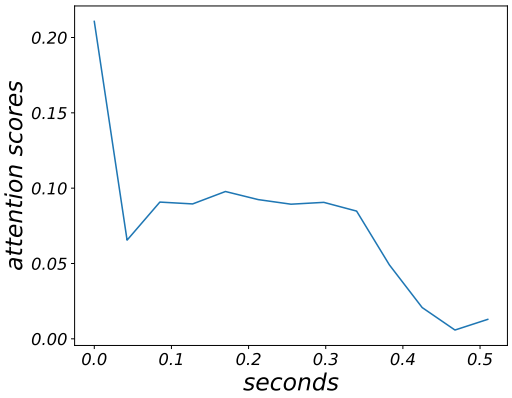


Figure 34: Attention score distribution of a clearing throat sound.

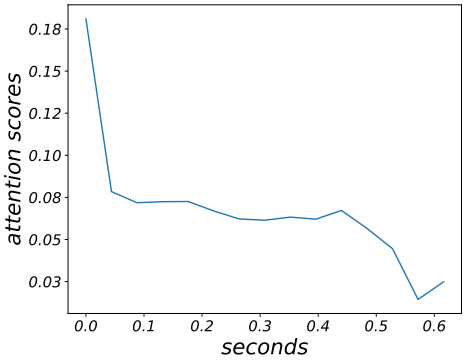


Figure 35: Attention score distribution of a lip smack sound.

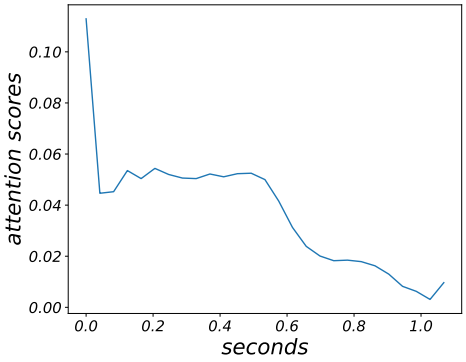


Figure 36: Attention score distribution of a sniff sound.

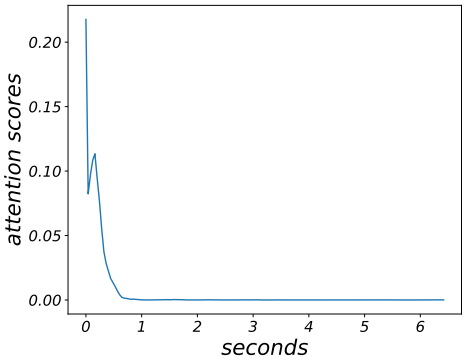


Figure 37: Attention score distribution of a breath sound.

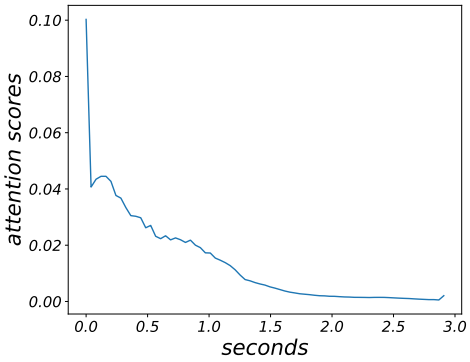


Figure 38: Attention score distribution of a laugh sound.

Since the extracted sounds have to be labeled manually (which is expensive), we decided to only extract the rarest sounds. Labeling 252 cough sounds would already double the number of labeled coughs in the training data. Cough, clearing throat, lip smack and sniff are the sounds that occur least frequently in the training data (see figure 16).
To compare different extraction methods, we manually labeled 100 sounds in the 'extraction-test-dataset'. Meaning that we listened to utterances where there was a sound in its transcript and transcribed its start and end time manually. We labeled all cough, clearing throat, and sniff sounds in the test dataset. Then we labeled lip smack sounds until we reached 100 sounds (see table 4). There were some utterances in the test dataset where we could not easily identify the start and end time of the transcribed sound. We omitted such sounds.

| sound | number of occurrences |
|---|---|
| lip smack | 66 |
| clearing throat | 21 |
| sniff | 8 |
| cough | 5 |

Table 4: Sounds in the 'extraction-test-dataset'.

**Max attention extraction:** When the BILSTM transcript of an utterance $u$ contains a sound $s$, we use the attention scores of the BILSTM to extract that sound from the original utterance audio. Let $x_1, ...., x_n$ be all hidden encoder states that correspond to $u$ and let $att[x_i]$ be the attention score of $x_i$ (the attention scores correspond to the sound $s$).

We extract sound $s$ by first computing $i_{max} = \arg\max_{i \in \{1,..,n\}}(att[x_i])$. Then, we determine the to $s$ corresponding start and end hidden encoder state in the following way ($k \in \mathbb{R}$):

$$i_{start} = min(\{i \in \{1, ..., i_{max}\} | \forall j \in \{i, ..., i_{max}\} : att[x_j] \geq \frac{att[x_{i_{max}}]}{k}\})$$
$$i_{end} = max(\{i \in \{i_{max}, ..., n\} | \forall j \in \{i_{max}, ...i\} : att[x_j] \geq \frac{att[x_{i_{max}}]}{k}\}).$$

Here, $\frac{1}{k} \cdot att[x_{i_{max}}]$ is the minimum attention score that is required for a hidden encoder state $x_i$ to still be considered part of the sound $s$. The start and end of the audio that corresponds to $x_{i_{start}}, ..., x_{i_{end}}$ is then used to extract the sound s from the utterance audio.

**Evaluation of the extraction method:** To determine the parameter $k$, we use the extraction-test-dataset to compare the extractions that are computed by different values of $k$. To automate this comparison, we need to compare the generated interval $[t'_{kstart}; t'_{kend}]$ of a specific value of $k$ with the correct interval $[t_{start}; t_{end}]$, where $t'_{kstart}, t'_{kend}, t_{start}$ and $t_{end}$ are timesteps in the utterance audio. It is desired to extract the whole sound, meaning that $[t_{start}; t_{end}] \in [t'_{kstart}; t'_{kend}]$. However, if $k \to \infty$ this would always be the case, since all attention scores are $\geq 0$. This means that sufficiently large $k$ extract the whole utterance and hence always contain the whole interval $[t_{start}; t_{end}]$. To reward large overlaps of $[t_{start}; t_{end}]$ and $[t'_{kstart}; t'_{kend}]$ and to punish large values of $k$, we introduce the *overlap_percentage* and the *extracted_length_percentage*:

$$[o_{start}; o_{end}] = \begin{cases} [t_{start}; t_{start}], & \text{if } [t_{start}; t_{end}] \cap [t'_{kstart}; t'_{kend}] = \emptyset \\ [t_{start}; t_{end}] \cap [t'_{kstart}; t'_{kend}], & \text{else} \end{cases}$$

$$extracted\_length\_percentage = \frac{o_{end} - o_{start}}{t'_{kend} - t'_{kstart}}$$

$$overlap\_percentage = \frac{o_{end} - o_{start}}{t_{end} - t_{start}}$$

The *overlap_percentage* and the *extracted_length_percentage* should both be as large as possible. We combine these two metrics to compute the score of an extraction which should be as close to 1 as possible:

$$score = overlap\_percentage \cdot extracted\_length\_percentage.$$

Since we want to evaluate the extraction method and not the model performance, we ignore sounds that are not detected. Figure 39 shows that $k = 5$ leads to the best average score.

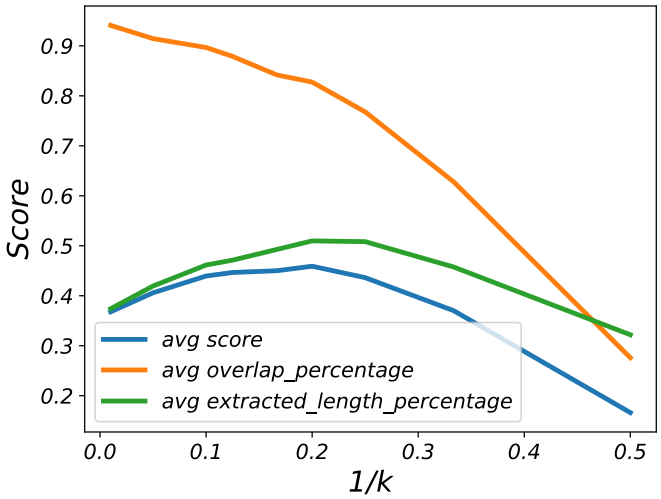


Figure 39: Average score of the max attention extraction method on the extraction-test-dataset.

### 5.4.2 Extracting and labeling sounds from other datasets

We used the max attention extraction method to extract cough, lip smack, throat clearing and sniff sound events from the Switchboard and Fisher dataset. We added 40 milliseconds of context before and after each extracted sound because otherwise some sounds were too short to be labeled manually. Each such extracted sound has been labeled by two persons on a website shown in figure 40. We created this website to maximize the efficiency of each labeling person. A person that wants to label an utterance can click on its timestamp to listen to the extracted sound. Then, the person that labels the sound has to choose the best fitting label. We added the labels unknown and nothing for extracted sounds that the labeling person can not classify or for extracted silence. Additionally, we added the hesitation class since hesitations like "mhhh" and "ahhhm" were often falsely extracted. Even though we only extracted the rarest sounds, we still included a laugh and breath label.

The external labeling process took around 3 months. This long delay between the data extraction and data labeling is one reason why data scarcity is a problem in sound event detection. Additionally, the external labeling of the data is expensive so it is not feasible to label unlimited amounts of new data for research purposes.

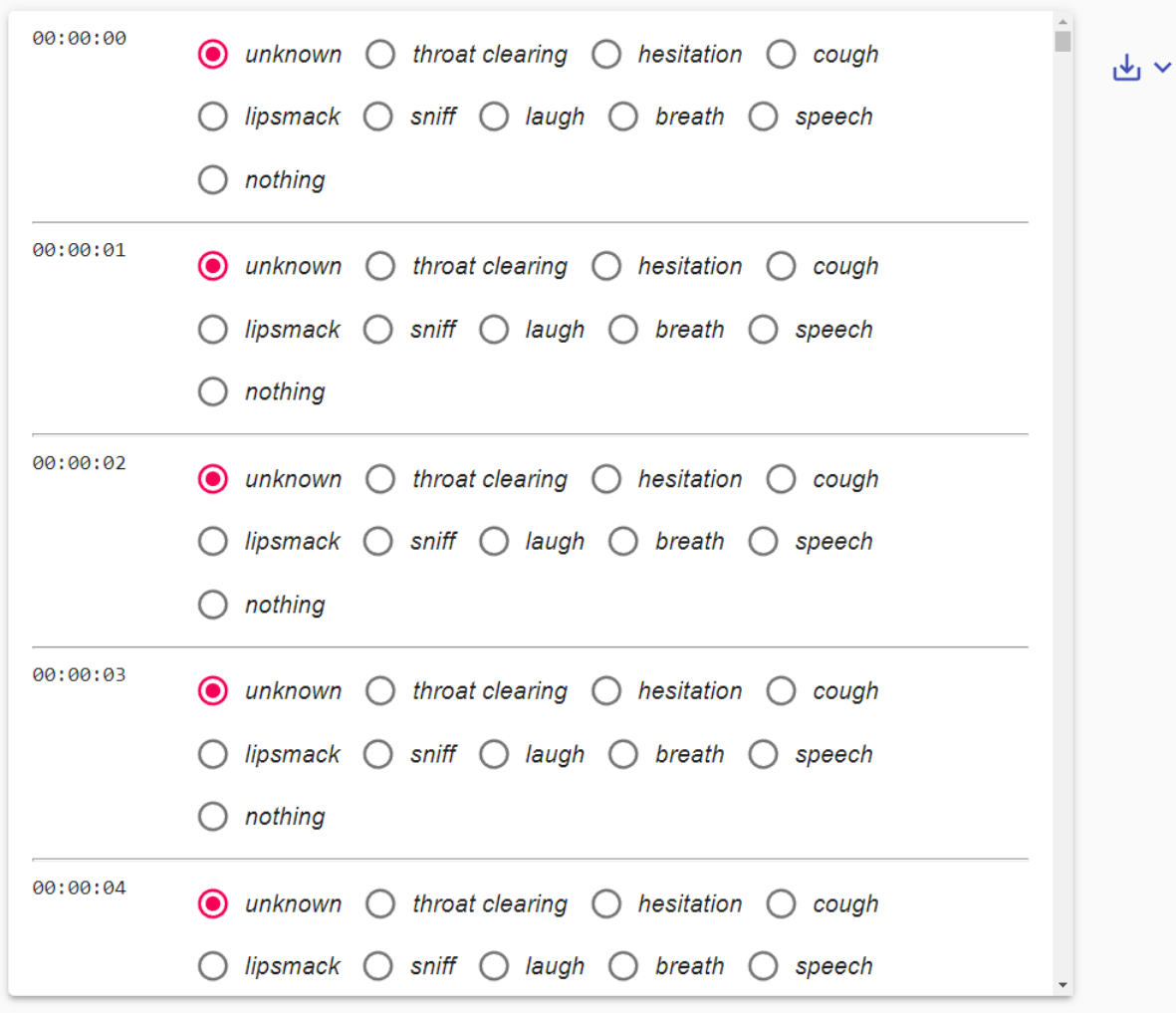


Figure 40: Labeling website.

It turns out that the people labeling sounds have a hard time assigning the correct label to short sounds. Short microphone noises are often labeled as lip smack sounds. This might be because the unknown class is not fitting for microphone noises that sound similar to a lip smack. The addition of a microphone noise class might resolve this issue. Another problem is that a lip smack sound often occurs at the beginning of a sentence when the speaker inhales. This inhaling sound event is sometimes falsely labeled as a sniff sound event. Even though listening to perfectly extracted sound events might be faster, providing the labeling persons with more context might increase the labels' quality. However, if too much context is extracted then the labeling person might be confused in cases where there are multiple sound events in one utterance.

### 5.4.3 Merging labeled sounds and the original transcript.

We keep only the sounds that have the same label assigned by both labeling persons. Then we compute an alignment as in the computation of the WER between the hypothesis and the transcript. When the manually labeled sound is labeled as an insertion and the surrounding words of the hypothesis are labeled as correct then we merge the labeled sound into the original transcript. This leads to new training data which distribution is shown in figure 41. Note that only the rarest sound events have been extracted and then labeled. Hence the low increase in breath, laugh and noise sounds.

The original sentence-level training data contains 252 cough sound events. This means that adding the new coughs to the training data increases the number of coughs in the training data by 80%. The addition of the newly labeled sniff sounds increases the number of sniffs by 118%. The increase in training data for all sounds is shown in table 5.

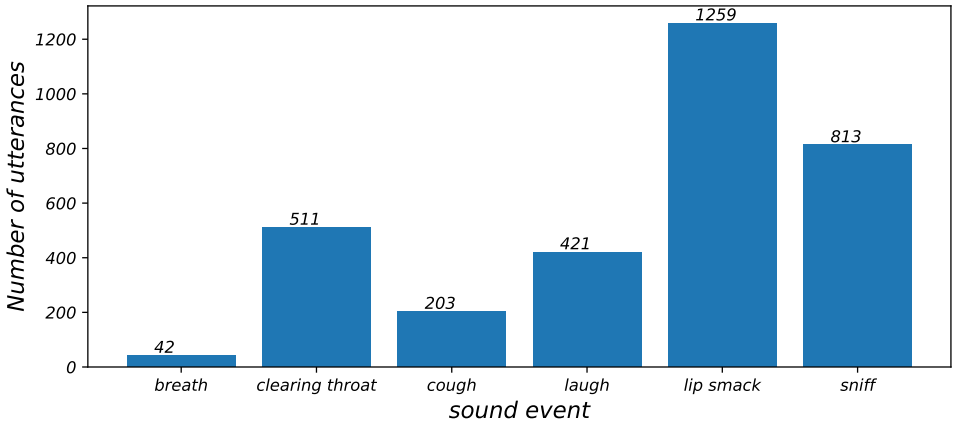


Figure 41: Number of utterances in the new labeled data from the Switchboard and Fisher dataset that contain a specific sound event.

| Sound event | Percentage increase |
|---|---|
| breath | 0.3% |
| clearing throat | 60.5% |
| cough | 80.6% |
| laugh | 4.1% |
| lip smack | 66.4% |
| noise | 0% |
| sniff | 118.6% |

Table 5: Increase in training data by adding the newly labeled data to the training dataset.

## 5.5 Training the finetuning model on the original and newly labeled sentence-level training data

We retrain the finetuning model on the original plus the newly labeled sentence-level training data to see if the new data increases the model's performance. Figure 42 shows that adding the new training data leads to very few changes. The WER of the model is shown in table 6. The model that was only trained on the original sentence-level training data has a WER of 16.3% while the new model has a WER of 16.4%.

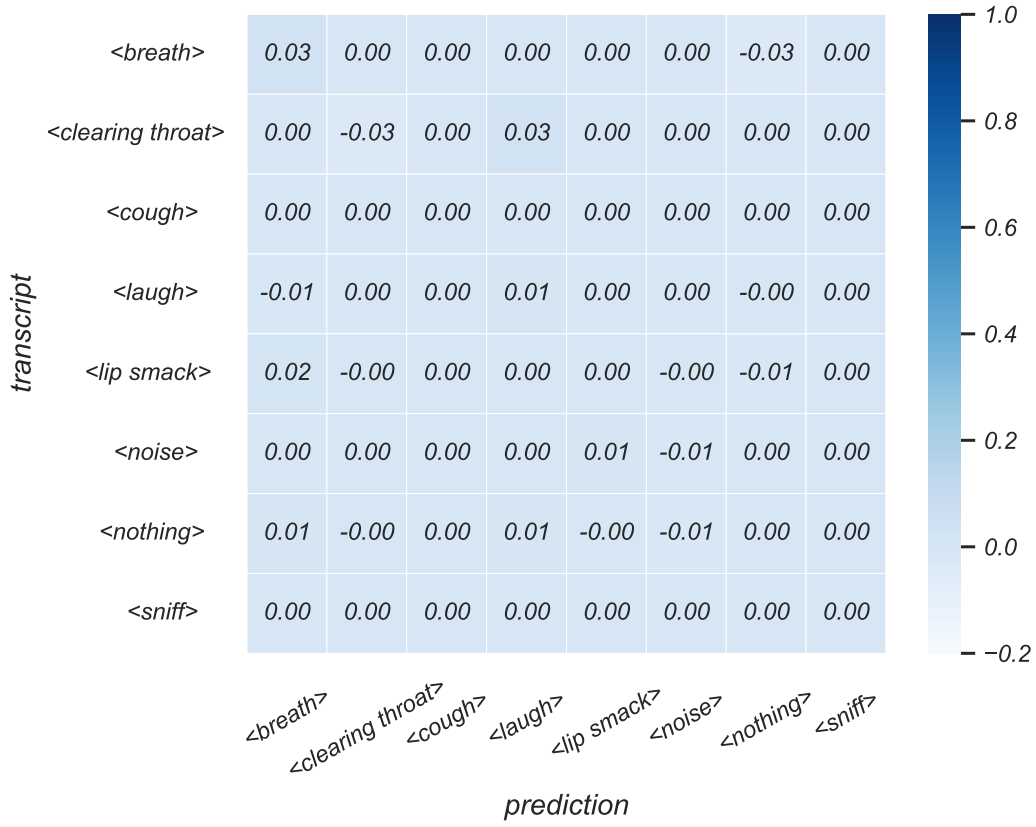


Figure 42: Improvement in sound event prediction accuracy by adding the newly labeled data to the sentence-level training dataset.

| Model | WER |
|---|---|
| Finetuning model | 16.4% |

Table 6: Word Error Rate of the finetuning model which is trained on the original sentence-level training data and the newly labeled data.

Finally, we also retrain the finetuning model by adding the newly labeled data to the sentence-level training dataset and then upsampling it. We upsample the data similarly to

the upsampled training data v1 dataset (see figure 43). Figure 44 shows the improvement of the system on the new upsampled training data.

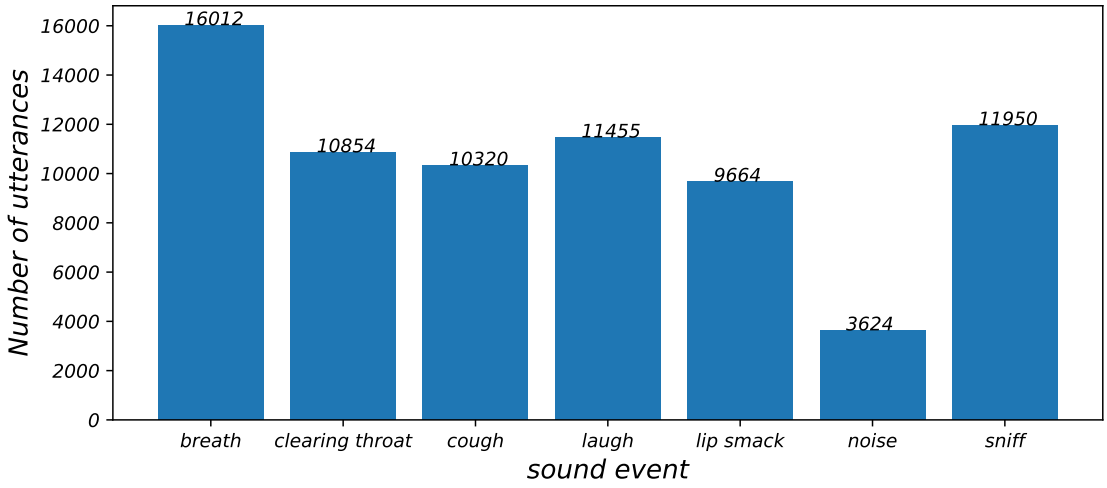


Figure 43: Number of utterances in the new upsampled sentence-level training data that contain a specific sound event.

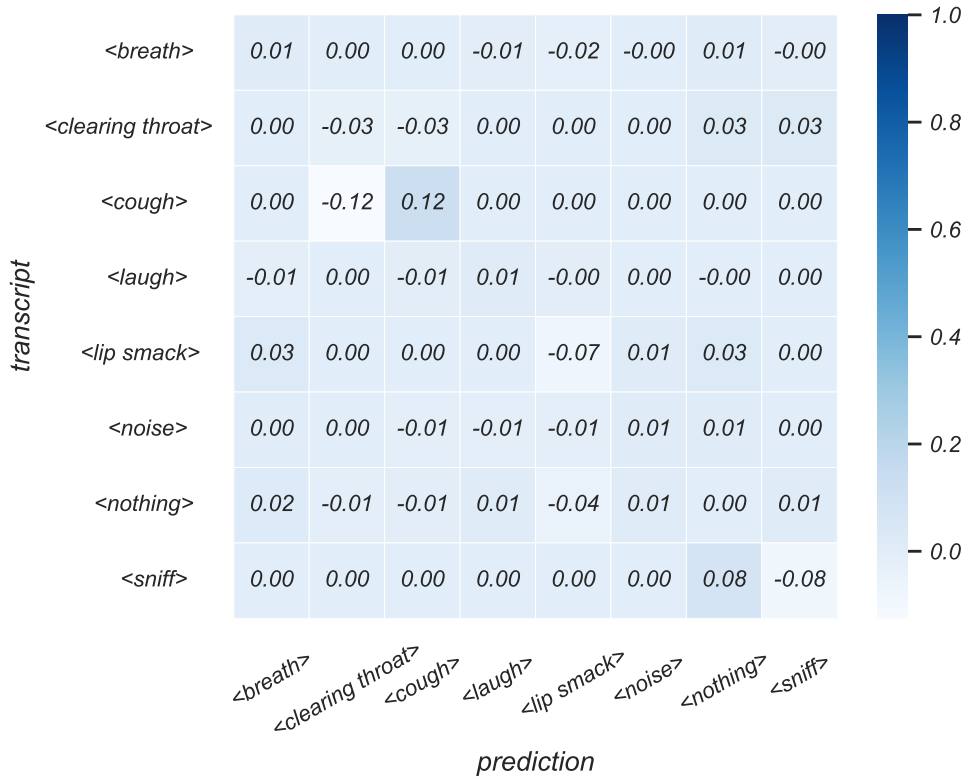


Figure 44: Improvement in sound event prediction accuracy by adding the newly labeled data to the sentence-level training dataset and upsampling it.

Table 7 shows that the WER did not change compared to the upsampled training data v1 dataset.

| Model | WER |
|---|---|
| Finetuning model | 16.7% |

Table 7: Word Error Rate of the finetuning model trained on the new upsampled training dataset.

Adding the manually labeled training data to the sentence-level dataset, before upsampling it, improves the classification of cough sounds by 12%. However, the classification accuracy of short sound events like lip smack and sniff decreased by 7% and 8% respectively. Listening to the new data shows that short sound events are more difficult for the labeling persons to classify. Some inhaling sounds were falsely classified as sniff and some microphone noises were falsely classified as lip smack. However, all the labeled cough sounds that we listened to were indeed cough sound events. This suggests that adding lower quality labels can decrease the system's performance while adding high quality labels increases the system's performance. As described in section 5.4.2, adding more context to short sound events or adding more labels to the labeling website might improve the label quality.

# 6 Review and Future Work

## 6.1 Review

We trained an attention based, bidirectional, sequence to sequence, long short-term memory model to generate transcripts from conversational speech that contain sound events. The sound events in our training data are: breath, clearing throat, cough, laugh, lip smack, noise and sniff sound events.

Upsampling the training data significantly increases the system's performance so that all sound events, except sniff, are detected and classified with over 70% accuracy.
Extracting and labeling new training data from the Switchboard and Fisher dataset did not change the model's WER, but increases the model's cough classification accuracy by 12%. However, adding the new training data also decreases the sound event classification accuracy of sniff and lip smack sound events by 8% and 7% respectively. Listening to the extracted sounds and comparing them to the assigned labels shows that the sound events sniff and lip smack are shorter and that their new labels are of lower quality than the other sound events. This lower quality data might be the reason for the decrease in sniff and lip smack sound event classification accuracy.

We also trained a word-level classifier which classifies words with 92% accuracy and performs significantly better on sniff sound events (from 54% accuracy to 84% accuracy). However, this model relies on the sentence being segmented into the individual words and sounds before inference.

## 6.2 Future Work

In order to increase the model's sound event classification accuracy, more utterances that contain sound events have to be labeled. Labeling new data by two labeling persons did not lead to very high quality labels for short sound events. Getting more persons to label the extracted sounds might lead to better labels. Also, the addition of a microphone noise label on the labeling website might increase the quality of the labels since some microphone noises have been falsely labeled as a lip smack sound event. Lastly, extracting sound events without context might not be sufficient for a labeling person to identify the correct label. One could use the best performing model to extract utterances that contain a sound event and have the labeling persons label whole utterances. This would ensure that, during the labeling process, the whole context is taken into consideration, but would increase the labeling time, and hence the cost, significantly.

Instead of training an end-to-end ASR system, one could also try to segment the input utterance into the individual sounds and words to use the better performing word-level classifier for sound event detection and classification. Then, one could merge transcribed sounds into a transcript that does not contain sound events. This would also allow the ASR model to be trained on conversational speech datasets that do not contain sound events.
Lastly, one could try optimization techniques that go beyond upsampling rare sound events to deal with the imbalances in the training data.

# Bibliography

[1] Ilham Addarrazi, Hassan Satori, and Khalid Satori. "Amazigh audiovisual speech recognition system design". In: *2017 Intelligent Systems and Computer Vision (ISCV)*. IEEE. 2017, pp. 1–5.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[3] Rui Cai et al. "Highlight sound effects detection in audio stream". In: *2003 International Conference on Multimedia and Expo. ICME'03. Proceedings (Cat. No. 03TH8698)*. Vol. 3. IEEE. 2003, pp. III–37.

[4] Steven Davis and Paul Mermelstein. "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences". In: *IEEE transactions on acoustics, speech, and signal processing* 28.4 (1980), pp. 357–366.

[5] Takashi Fukuda, Osamu Ichikawa, and Masafumi Nishimura. "Detecting breathing sounds in realistic Japanese telephone conversations and its application to automatic speech recognition". In: *Speech Communication* 98 (2018), pp. 95–103.

[6] Philip Gage. "A new algorithm for data compression". In: *C Users Journal* 12.2 (1994), pp. 23–38.

[7] Jort F. Gemmeke et al. "Audio Set: An ontology and human-labeled dataset for audio events". In: *Proc. IEEE ICASSP 2017*. New Orleans, LA, 2017.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[9] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[10] Xuedong Huang et al. *Spoken language processing: A guide to theory, algorithm, and system development*. Prentice hall PTR, 2001.

[11] Yakubu A Ibrahim, Juliet C Odiketa, and Tunji S Ibiyemi. "Preprocessing technique in automatic speech recognition for human computer interaction: an overview". In: *Annals. Computer Science Series* 15.1 (2017), pp. 186–191.

[12] Hirofumi Inaguma et al. "An end-to-end approach to joint social signal detection and automatic speech recognition". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 6214–6218.

[13] Daniel Jurafsky and James H. Martin. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition*. December 30, 2020.

[14] Lyndon S Kennedy and Daniel PW Ellis. "Laughter detection in meetings". In: (2004).

[15] Mary Tai Knox and Nikki Mirghafori. "Automatic laughter detection using neural networks." In: *Interspeech*. Citeseer. 2007, pp. 2973–2976.

[16] Taku Kudo and John Richardson. "Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing". In: *arXiv preprint arXiv:1808.06226* (2018).

[17] Richard Lippmann. "An introduction to computing with neural nets". In: *IEEE Assp magazine* 4.2 (1987), pp. 4–22.

[18] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. "Effective approaches to attention-based neural machine translation". In: *arXiv preprint arXiv:1508.04025* (2015).

[19] Nelson Morgan et al. "The meeting project at ICSI". In: *Proceedings of the first international conference on human language technology research*. 2001.

[20] Thai-Son Nguyen et al. "Improving sequence-to-sequence speech recognition training with on-the-fly data augmentation". In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 7689–7693.

[21] Prajit Ramachandran, Barret Zoph, and Quoc V Le. "Searching for activation functions". In: *arXiv preprint arXiv:1710.05941* (2017).

[22] Boris Reuderink et al. "Decision-level fusion for audio-visual laughter detection". In: *International Workshop on Machine Learning for Multimodal Interaction*. Springer. 2008, pp. 137–148.

[23] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[24] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[25] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[26] Stefan Scherer et al. "Multimodal laughter detection in natural discourses". In: *Human Centered Robot Systems*. Springer, 2009, pp. 111–120.

[27] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[28] Khiet P Truong and David A van Leeuwen. "Automatic detection of laughter". In: *Ninth European Conference on Speech Communication and Technology*. 2005.

[29] Alex Waibel et al. "Phoneme recognition using time-delay neural networks". In: *IEEE transactions on acoustics, speech, and signal processing* 37.3 (1989), pp. 328–339.

[30] Li Wan et al. "Regularization of neural networks using dropconnect". In: *International conference on machine learning*. PMLR. 2013, pp. 1058–1066.

[31] Xinghuo Yu, M Onder Efe, and Okyay Kaynak. "A general backpropagation algorithm for feedforward neural networks learning". In: *IEEE transactions on neural networks* 13.1 (2002), pp. 251–254.

[32] Yong Yu et al. "A review of recurrent neural networks: LSTM cells and network architectures". In: *Neural computation* 31.7 (2019), pp. 1235–1270.

# A Appendix

## A.1 Evaluation

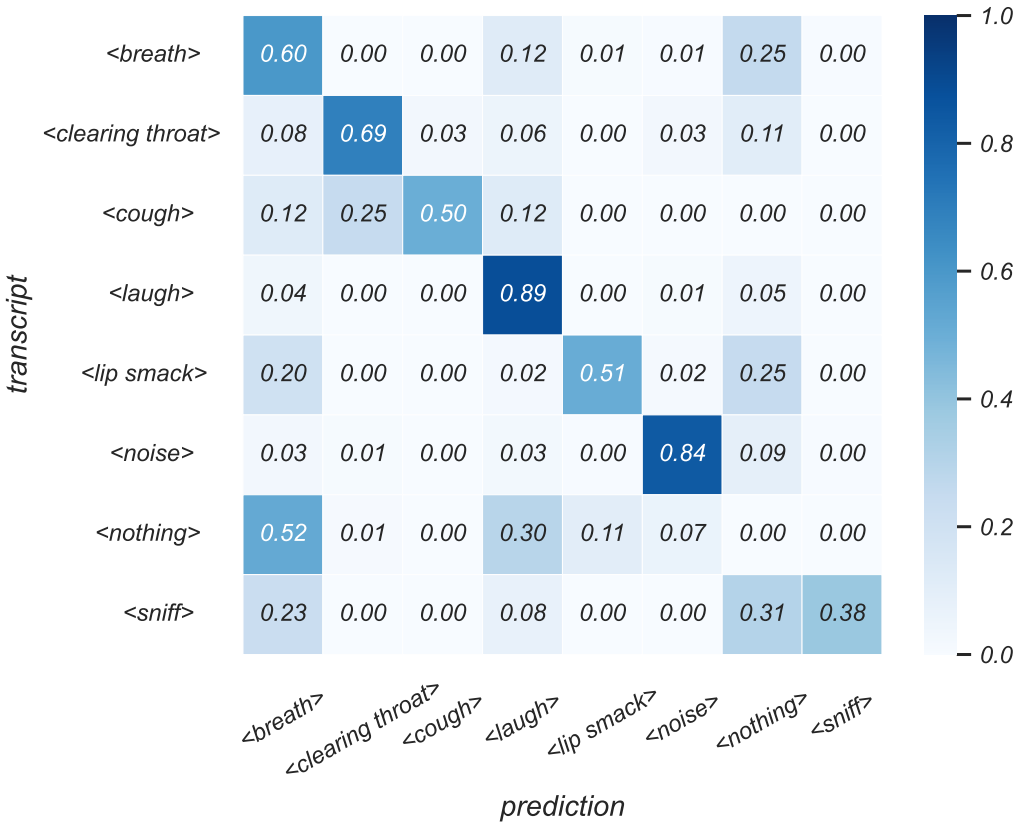


Figure 45: Sound classification accuracy of the model that is trained from scratch on the sentence-level training data.

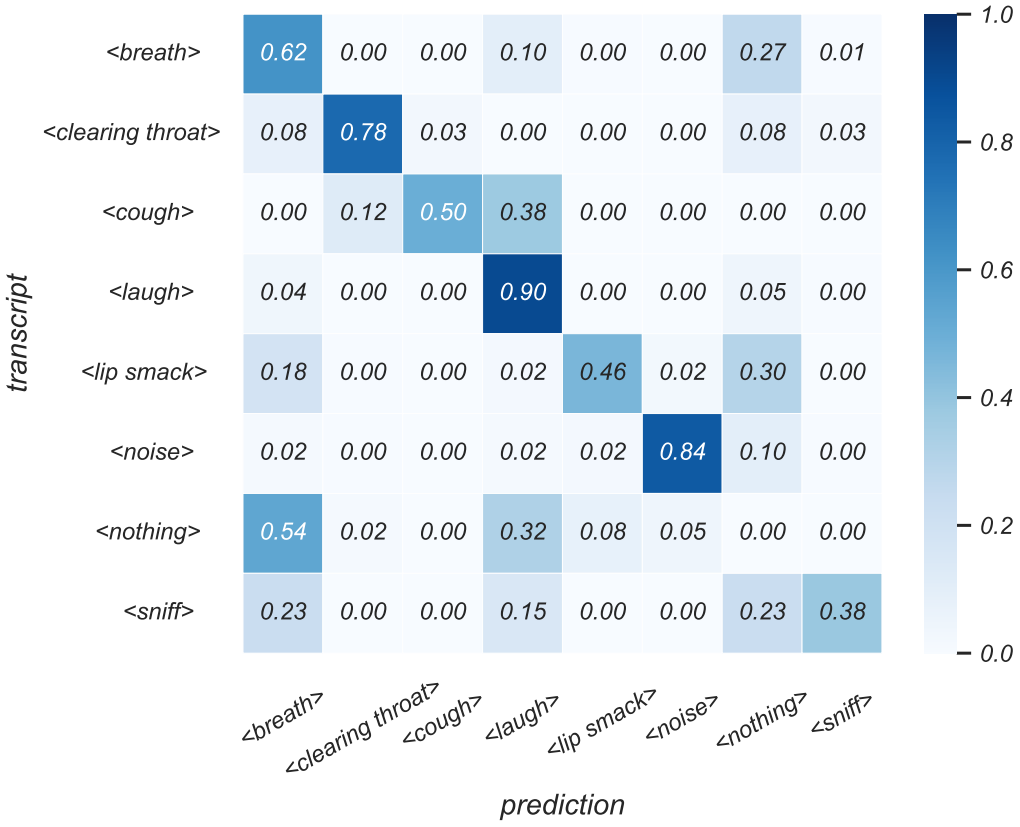


Figure 46: Sound classification accuracy of the finetuning model with a frozen decoder that is trained on the sentence-level training data.

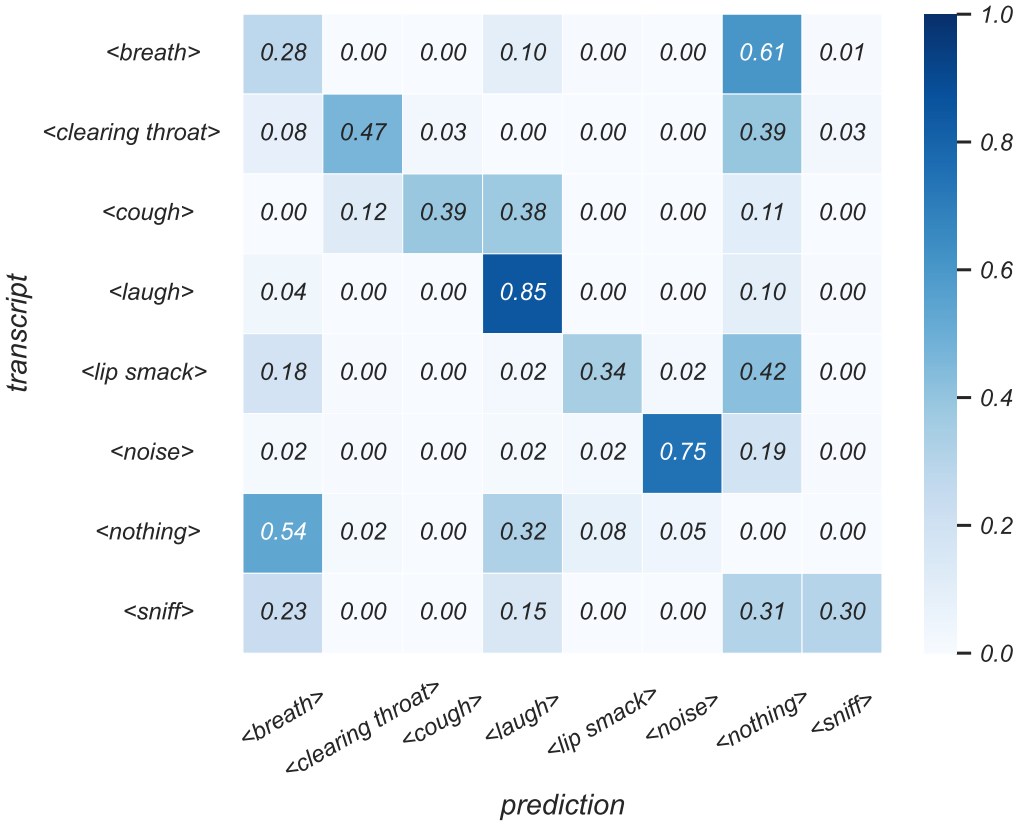


Figure 47: Sound classification accuracy of the finetuning model with a frozen encoder which is trained on the sentence-level training data.

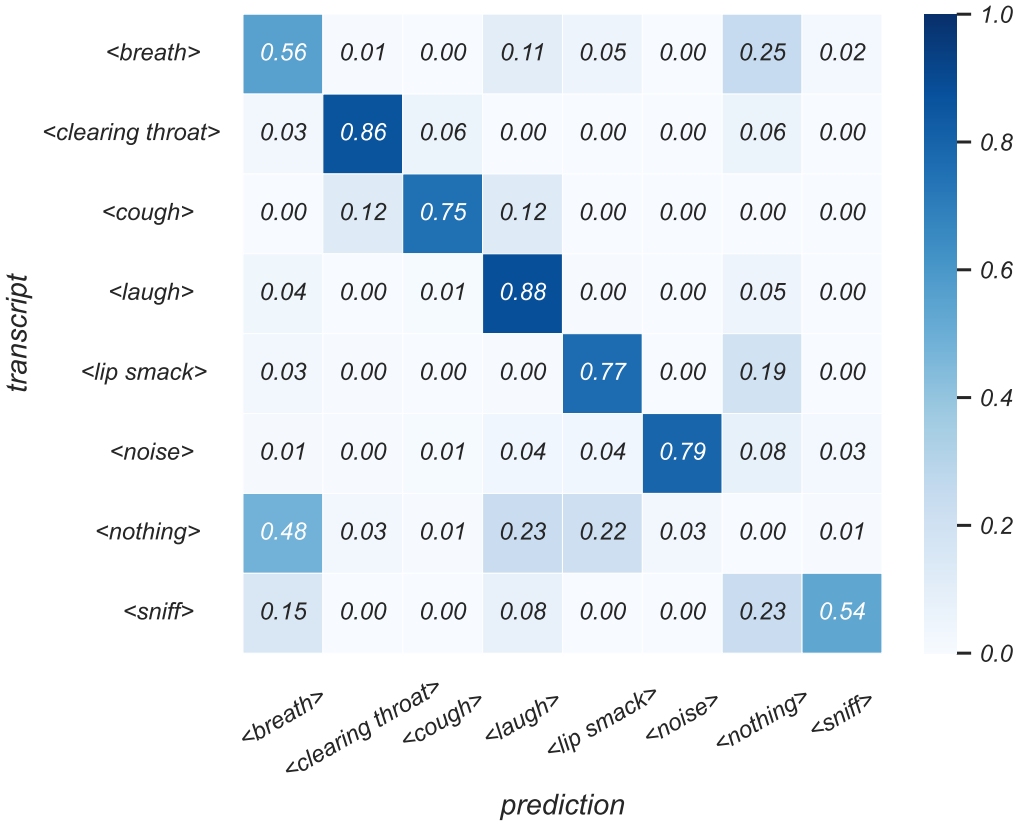


Figure 48: Sound classification accuracy of the finetuning model which is trained on the upsampled training data v1 dataset.

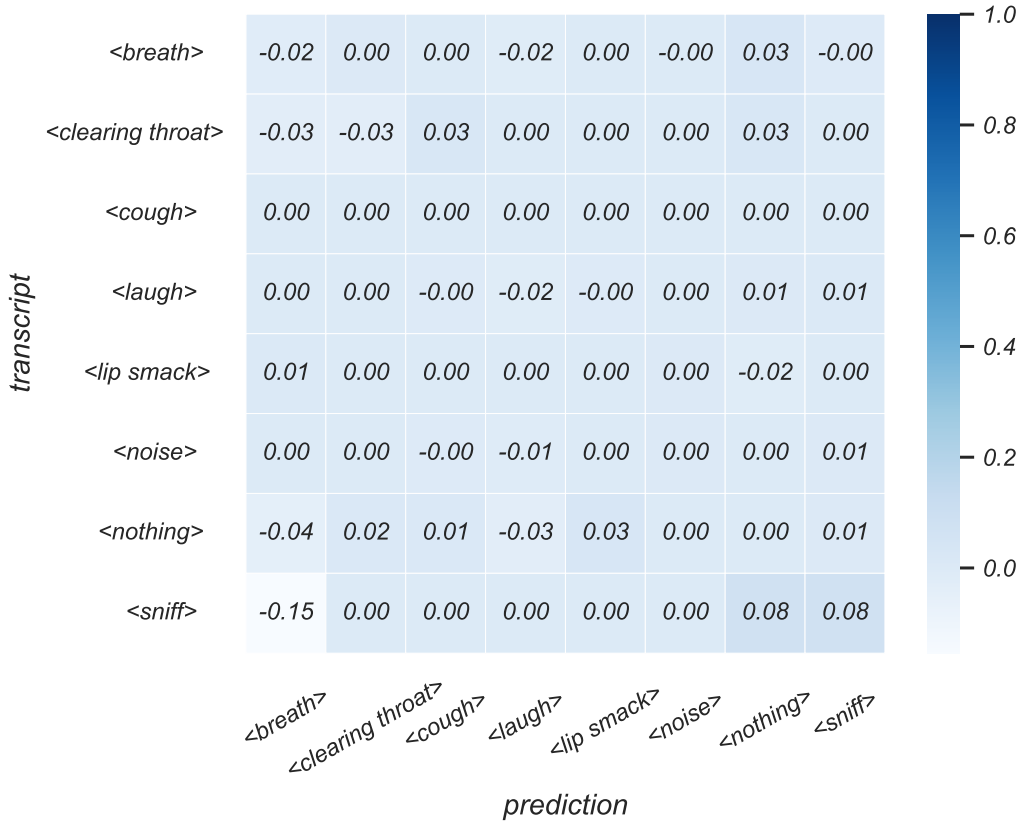


Figure 49: Improvements by doubling the amount of sniffs in the upsampled training data v1 dataset.

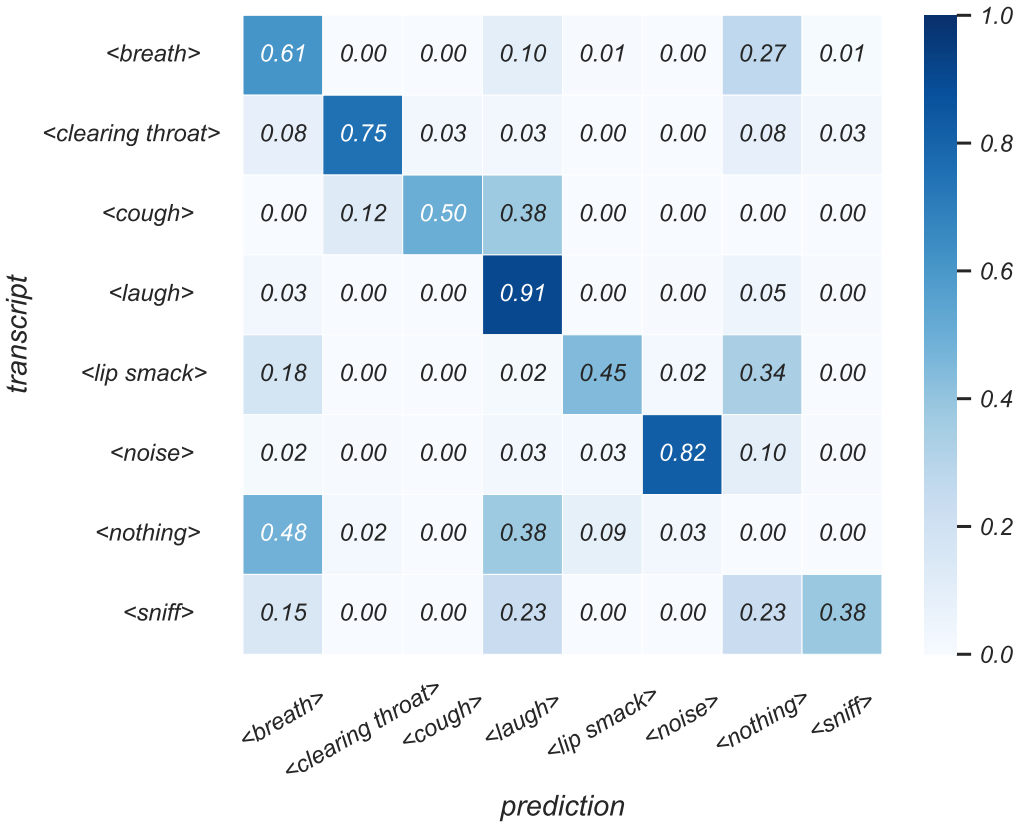


Figure 50: Sound classification accuracy of the finetuning model which is trained on the dataset that is obtained by adding the newly labeled data to the sentence-level training dataset.

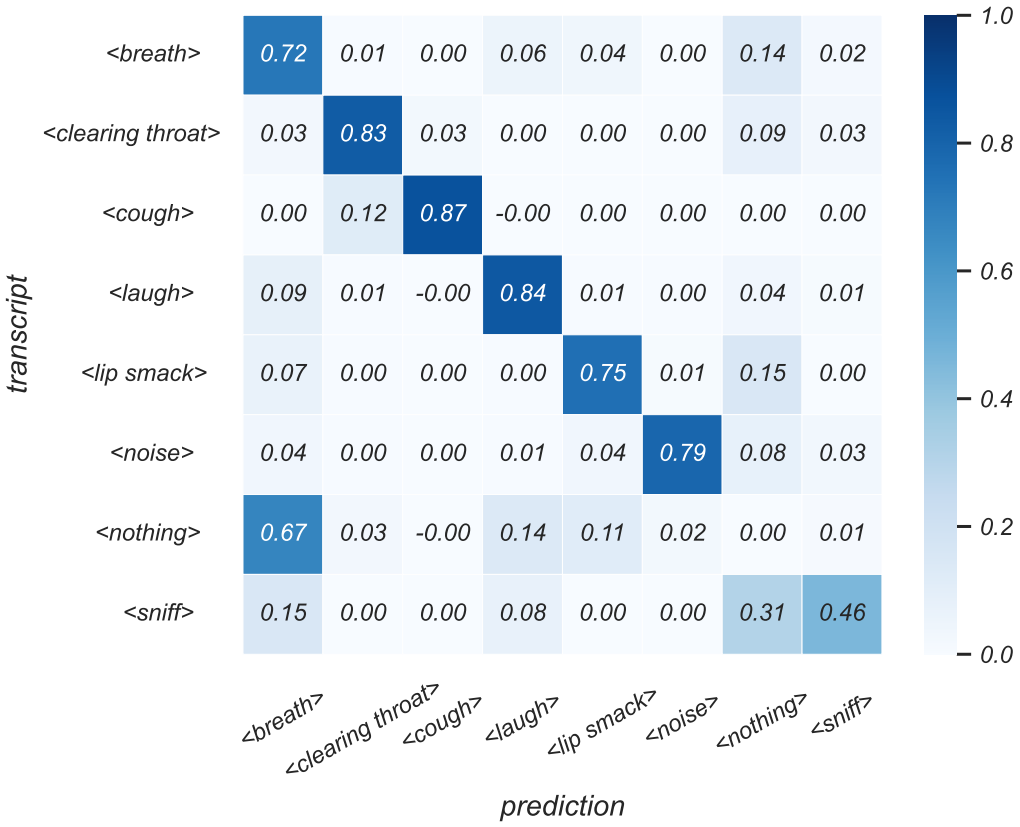


Figure 51: Sound classification accuracy of the finetuning model which is trained on the dataset that is obtained by adding the newly labeled data to the sentence-level training dataset and upsampling it.

# Assertion

*Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.*

Karlsruhe, September 1, 2022

*Stefan Meintrup*