Bachelor thesis
Computing Science

Radboud University

Fine-tuning Wav2vec2.0 on caption data

Author:
Thomas Kolb
s1027332

First supervisor/assessor:
Professor, David van Leeuwen
d.vanleeuwen@science.ru.nl

Second assessor:
Doctor, Louis ten Bosch
louis.tenbosch@ru.nl

January 19, 2022

Abstract

Interpreting spoken audio to create captions is a time-consuming process. Yet 1.5 million deaf and hard hearing in the Netherlands are in need of it every day. Innovations in automatic speech recognition give rise to new research opportunities related to generating captions automatically. In this research, we aim to find out whether we can use caption data written by NPO caption interpreters to train Wav2vec2 to automate the process of writing captions. In addition, we will explore the capabilities of Wav2Vec2 when learning properties of captions.

In our experiments, we find that Wav2vec2 can capture caption properties such as interpunction and summarization quite well. We explored the impact of using different Wav2vec2 model variants and how it influences the performance. In addition, we found that the application of certain preprocessing techniques can influence the quality of learning significantly.

The scripts we wrote and the experiment results can be found on our Github repository: `https://github.com/Thomaskolb/fine-tuning-w2v2`

Contents

Chapter 1

Introduction

There are different ways to transcribe spoken language to text. We can choose to be very literal and interpret every word that is said directly to text, i.e., verbatim transcription. If the goal is to understand what somebody is trying to convey then transcribing this literally might not be the best approach. We are not used to reading verbatim transcriptions and people are limited by their ability to speak a language correctly, their grammatical knowledge and vocabulary. Such limits can mislead the person who is trying to understand something. This problem is especially apparent when the person watching is deaf or hearing impaired and does not have the auditory support to puzzle together the idea that is being displayed. When creating closed captions for television the human transcriber tries to avoid this.

The human transcribers' task is to try and understand what is being said and transcribe it in such a way that it is easier to understand for the people watching and reading the captions. This could mean summarizing, fixing grammatical errors, adding interpunction or capitalization for readability. It also happens that important information is lost when speech is translated into text. For example question marks suggest that the sentence is a question, this information is usually lost when speech is transcribed to verbatim text. Below we display an example of the discrepancy of closed captions (also meaning-for-meaning) and verbatim text, as demonstrated in an article about speech to text services by the National Deaf Center [10]:

Verbatim:

```
-Darcy you deal in stocks, don't you.
-Yes, I do.
-I just want to give you this little thing I heard
recently, you know, that if you have stock in Continental
Bank, you know, get it out really quick because they are
about to go bankrupt. Okay. Got that. Good. Good.
```

Meaning-for-meaning:

```
-Professor: Darcy, you deal in stocks, don't you?
-Darcy: Yes, I do.
-Professor: I wanted to give you a tip I heard recently. If
you have stock in Continental Bank, get out now. They are
about to go bankrupt. Got that?
```

Clearly closed captions can hold a lot of information that is usually lost when transcribing speech, as well as removing redundant text that holds no additional meaning and would only serve to distract readers. Creating captions is a very time-consuming process, it makes sense to consider automatizing this process.

In this research, we will train the end to end model Wav2vec2 [6] using audio .WAV data and their corresponding captions from the Dutch broadcast establishment NPO (Nederlandse publieke omroep). We think it is important to research the automatization of transcribing captions. The reason we used data from NPO is that human transcribers from NPO transcribe captions under strict and consistent guidelines (in section 3.1.4 we examine these guidelines in more detail). This is an important property to make the training process easier for the model. In this research we try to answer the following question:

- *How well does Wav2vec2, trained on captions, transcribe captions from spoken audio compared to Wav2vec2 trained on verbatim text labels?*

We essentially try a different approach for generating captions from using the data that is usually used to train an ASR (Automatic speech recognition) model. Another focus of this research is to see how the results compare, we try to push the limits of the Wav2vec2 model to see if is capable to learn certain properties of captions that verbatim text lacks. Another important part of this research is to find out whether we can use the start- and endtime which are provided with the caption text to train Wav2vec2 to properly transcribe captions.

- *What properties of NPO captions can be captured by Wav2vec2 when using captions as labels for the training?*
- *How useful is the raw caption data provided by NPO when training Wav2Vec2 to transcribe captions? What amount of pre-processing is required?*

1.1 Contributions

Wav2vec2 is a very promising model in NLP. It could be described as the audio version of BERT [15] (BERT is a text-only NLP model which was very innovative due to the introduction of bidirectional representations) Wav2vec2 performs very well on low amounts of labeled data [6]. Large amounts of labeled data are often difficult to get by, especially for less widely spoken languages. It is therefore important that we investigate the limitations of Wav2vec2. We build on related work on the model, which examined the ability to perform speaker recognition [24], and performance of automatic speech recognition on low-resource languages [26]. This study will investigate the limits of Wav2vec2 when learning closed captions. More formally:

- We investigate the current state of Wav2Vec2 when transcribing captions using captions as labels.
- We explore the ability for Wav2vec2 to learn certain properties of closed captions.
- We try to find out under what circumstances closed captions are learned best. i.e., what kind of pre-processing is required.

Chapter 2

Preliminaries

2.1 Artificial neural network

To get an understanding of the inner workings of Wav2vec2 [6] we need to get an intuition of the basics. Artificial neural networks (ANNs) are a subset of machine learning, present at the core of every deep learning model. Neural networks are built on the concept of *linear regression*.

2.1.1 Linear regression

The goal of regression [17] is to *characterize* the relation between inputs and outputs. It is used to predict the value of a variable based on the value of other variables. We will call the input vector $\mathbf{x}$ of length n containing values for different attributes of the input, which corresponds to the output variable y. To indicate a prediction of y we will use $\hat{y}$. In a *linear model* the assumption of linearity makes it possible to express y such that:

$$\hat{y} = w_1 * x_1 + ... + w_n * x_n + b$$

Or:

$$\hat{y} = \mathbf{x}\mathbf{w} + b$$

Here the vector $\mathbf{w}$ are the weights, which basically say for each variable of the vector $\mathbf{x}$ how important the variable is to predict y. For example, when predicting the edibility of mushrooms, the color of the mushroom might tell us more than the number of gills. The bias b will add some value that was needed to predict y properly when all the variables of $\mathbf{w}$ would take the value of 0. This process is called *linear regression*. In a neural network, the goal is to modify the weight and bias values such that our prediction $\hat{y}$ would be closest to y.

If we want to increase the complexity of the network we will have to deviate from linear regression by introducing more *hidden layers*. Each hidden

layer is essentially 'predicts' the values of the next layer with the use of another **w** vector, or the values of output. Depending on the shape of this **w** vector we can control the number of output values. This is visualized in Figure 2.1. In this example the linear relation between the input layer and the first hidden layer looks like this:

$$\mathbf{h} = \mathbf{x}\mathbf{w} + b$$

Often an *activation function* ϕ is used, this can have different functions. *Softmax activation* [7] for example normalizes the value to a range between -1 and 1.

$$\mathbf{h} = \phi(\mathbf{x}\mathbf{w} + b)$$

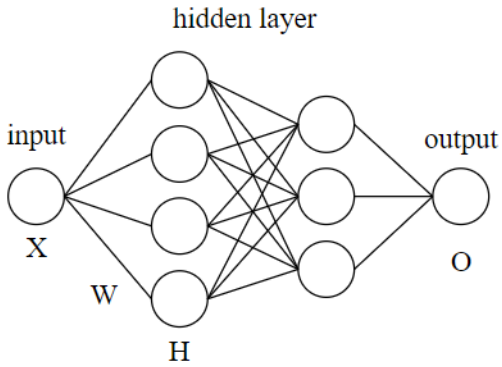

Figure 2.1: Neural network structure

2.1.2 Loss function

To measure the *fitness* of our model parameters we introduce the loss function. The loss function quantifies the distance between real value y and our prediction $\hat{y}$ of a given sample. An example loss function for a given sample i is *quadratic loss*:

$$l^{(i)}(\mathbf{w}, b) = \frac{1}{2}(\hat{y}^i - y^i)^2$$

For an entire dataset with k samples we average out the loss:

$$L(\mathbf{w}, b) = \frac{1}{k}\sum_{i=1}^{k} l^{(i)}(\mathbf{w}, b)$$

2.1.3 Training process

The goal of the training process is to change our parameters $\mathbf{w}$ and b such that we minimize the loss function (the objective function). We do this by iteratively updating the parameters in the direction that lowers the loss function. We can calculate this direction by taking the partial derivative of the loss function concerning the $\mathbf{w}$ and b parameters. In each iteration we apply the following formula:

$$(\mathbf{w}, b) \longleftarrow (\mathbf{w}, b) - \eta * \partial_{(\mathbf{w},b)} L(\mathbf{w}, b)$$

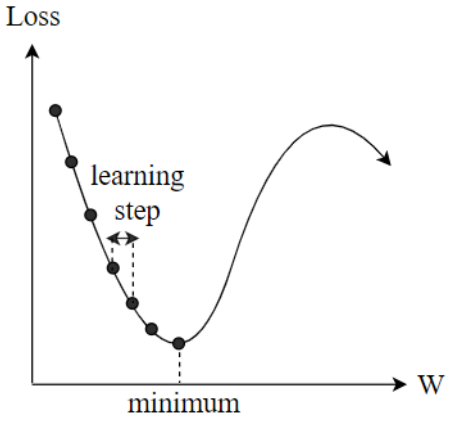

Figure 2.2: Gradient descent

This process is called *gradient descent* [16] (GD), shown in Figure 2.2. The learning rate η determines the step size at each iteration, having a large learning rate would mean that we converge to a minimum faster, but this also has the risk that we skip the minimum because we went too far. Finding the right value for η is a difficult task that strongly depends on the specifics of the problem. In practice passing over the entire dataset and calculating the partial derivatives to update w and b is very expensive. One solution that is often used is *minibatch stochastic gradient descent* (MSGD) with $minibatch = \beta$:

$$(\mathbf{w}, b) \longleftarrow (\mathbf{w}, b) - \frac{\eta}{|\beta|} \partial_{(\mathbf{w},b)} l^{(i)}(\mathbf{w}, b)$$

In *stochastic gradient descent* [20] (SGD) we only use a *single* random sample to update the $\mathbf{w}$ and b. In MSGD we take a *subset* of the entire dataset to update the parameters. SGD converges much faster compared to GD, one disadvantage is that the values of $\mathbf{w}$ and b oscillate due to its sensitivity to

noise. MSGD does not have this disadvantage (because we take the average change of a set of samples) and still converges a lot faster compared to GD, even though it is slightly slower than SGD.

2.2 Deep learning concepts

To explain the inner workings of Wav2vec2 better, we need to explain some core concepts within deep learning before we can have some understanding of what is happening with our data in this study.

2.2.1 Sequential data

In this research, we work with short audio fragments (as input) and brief caption texts (as labels). Both these forms of data are sequential. Sequential data means that the points in the dataset are dependent on other points in the dataset. It is important to make this distinction between 'normal' data because sequential data tends to be an issue for the traditional neural network. This makes a lot of sense because when we look at Figure 2.1 it is clear that each input sample is analyzed in isolation from any of the other input samples. This template is limited to data where the samples only depend on the attributes of themselves.

2.2.2 Recurrent neural network

To solve this issue we need to make a change to the structure of the traditional neural network. We need to make sure that data points can depend on other data points. Previously we discussed that the linear relation for a traditional neural network between input and a hidden layer is:

$$\mathbf{h} = \phi(\mathbf{x}\mathbf{w} + b)$$

In an RNN [21] (based on [12]) we need to make sure that information from previous input x_{t-1} is used in the calculation of the current input x_t. We can make sure that this happens when we try to implement the following linear relation in the structure of the network where h is a hidden unit:

$$h_t = f(x_t, h_{t-1})$$

For the hidden unit at time t we use the information of the current input x_t and the previous hidden unit h_{t-1} (which contains information of x_{t-1}, thus the previous input). The linear relation between the input vector X and a hidden layer H in an RNN is the following:

$$\mathbf{h} = \phi(\mathbf{x}_t\mathbf{w} + \mathbf{h}_{t-1}\mathbf{w} + b)$$

The structure is visualized in Figure 2.3

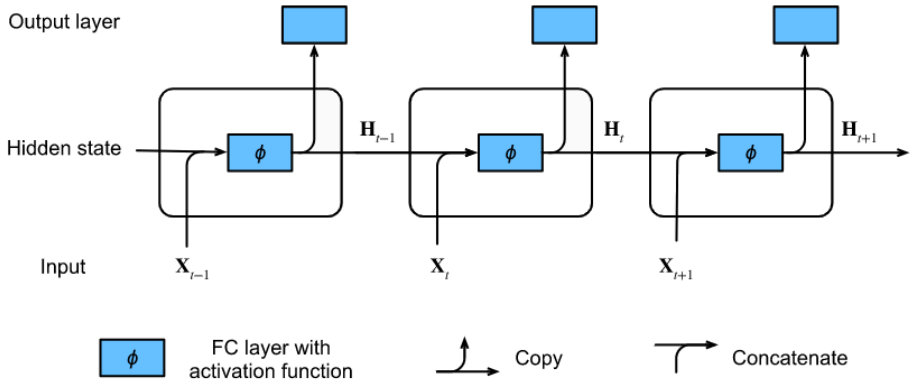

Figure 2.3: Recurrent neural network [17]

2.2.3 Attention mechanisms

In natural language processing, the key idea is to train a network such that it gains the ability to understand text or spoken words. In an RNN one could argue that the attention when learning for example a sequence of words is on the previous words, where the last word gets more attention than the words that came before. Let's look at the following example:

```
Stephen is very happy
```

Let's say the RNN is considering the word happy, then it will know that the word that came before `very` is relevant to the word happy, the word `Stephen` is much less important according to this model. By introducing *attention mechanisms* we can solve this issue. We take the input as a vector of n tokens $\mathbf{t}$, for this example:

$$\mathbf{t} = [Stephen, is, very, happy]$$

In *self-attention* [8] the goal is to give these tokens (or *embeddings*) more *context*, this means we summarize in each token how they relate to the other words in the sentence. An important part of attention and also transformers [13] are *keys* $\mathbf{k}$, *values* $\mathbf{v}$ and *queries* $\mathbf{q}$. The following matrix multiplications are performed to get these vectors:

$$\mathbf{v} = \mathbf{t} * M_v$$

$$\mathbf{k} = \mathbf{t} * M_k$$

$$\mathbf{q} = \mathbf{t} * M_q$$

The *query* q could be interpreted as the token we are currently considering (where $\mathbf{q}$ is the set of queries q for all tokens $\mathbf{t}$), for example, `happy`. The

query and keys are combined using an attention scoring function one by one, this could be for example the *dot-product*. The output of this could be described as the *attention weights*, for the combination (`Stephen`, `happy`) this value will likely be very high. Next, we combine each weight with the *values* **k**, the resulting output will be the best-matched locations of where to pay attention. Figure 2.4 Visualizes this process more clearly.

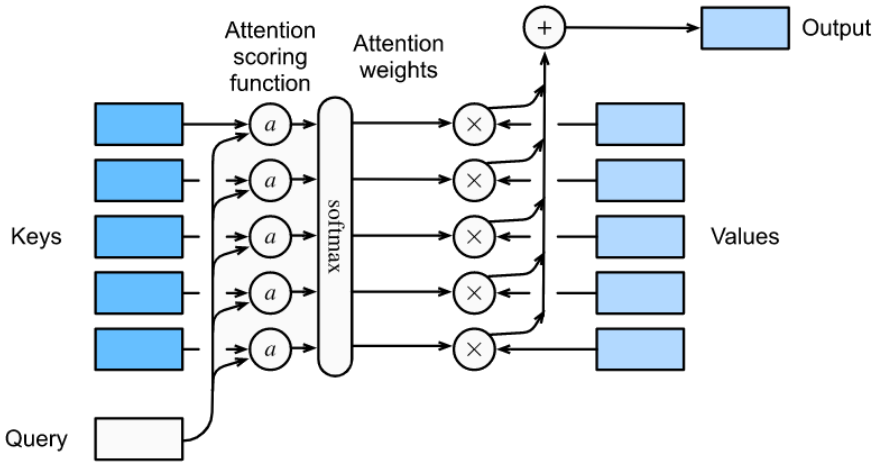

Figure 2.4: Attention mechanism [17]

The matrices M_v, M_k and M_q will be our training parameters. Because we multiply the tokens T with three different matrices there is a lot of room for expressiveness to describe the data. To add more information we can also choose to inject *positional encodings* to the input. This will allow is to use the *sequence order information*.

In *multi-head attention*, we run this exact process of *self-attention* several times in parallel. Then in the end we concatenate these independent attention outputs and normalize the data. This allows us to capture more dependencies between the tokens, essentially more expressiveness.

2.2.4 Transformer layer

In a transformer network [13] we combine the previously introduced concepts. The model architecture of the transformer is visualized in Figure 2.5. In short, a transformer consists of an *encoder* and *decoder* block. When training an ASR the encoder will take the audio as input, with the injection of their *positional encodings*. Next, we apply *multi-head attention* to capture

the dependencies. It is important to note that we can learn these dependencies for each embedding in parallel. Lastly, we use an FFN (feed-forward network) to transform the attention vectors into a form that can be used by the next layer. The labels, which will be text for an ASR will be used as input for the decoder. Similar to the encoder, in the decoder, we will also inject its *positional encodings*. Next, we apply *masked multi-head attention*. Here it tries to predict the next text label, the *masked* part refers to the text labels that come after which are hidden to the network. This is because we need to predict the next label in order to learn. Next we apply another layer of *multi-head attention*, this will also take the output from the *encoder*. This is where the transformer tries to map the audio embeddings to text labels and figures out what the relation between them is. The last layer is another FFN, which makes sure that the vectors can be easily accepted by another neural network layer.

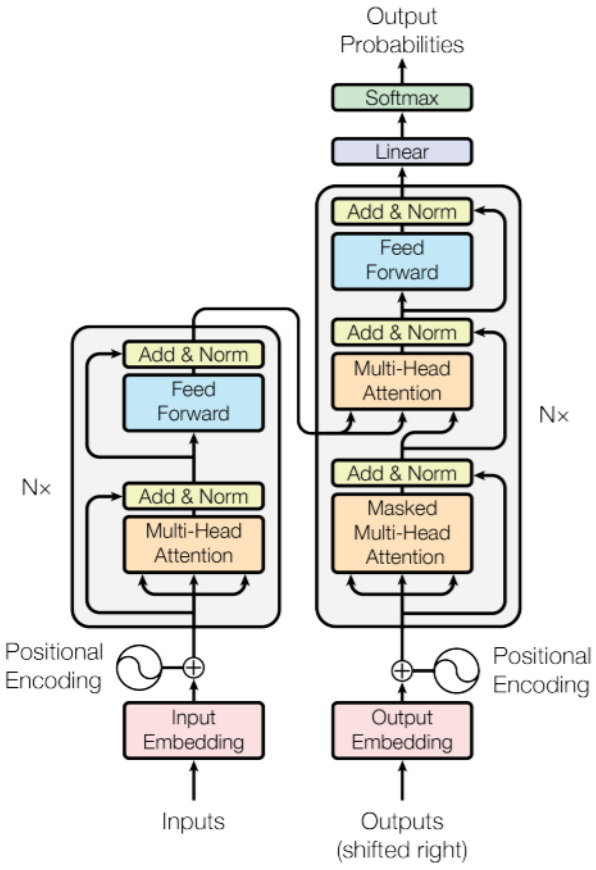

Figure 2.5: Transformer architecture [13]

2.2.5 Convolutional layer

We can use *convolutional layers* [14] to increase or decrease the size of the input vector. The size of certain input data can grow very large quickly considering we need weights for each input unit. Convolutional layers are often used to decrease the size of *image data*. In Figure 2.6 we decreased the size of a 3 by 3 image to 2 by 2. We do this using a *kernel*.

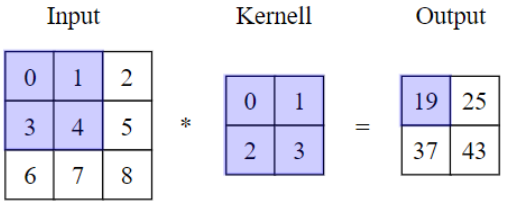

Figure 2.6: Convolution [17]

We can determine the output size $o_h \times o_w$ of the image in the following way:

$$(o_h \times o_w) = ((n_h - k_h + 1) \times (n_w - k_w + 1))$$

Where the input size is $n_h \times n_w$ and the kernel size is $k_h \times k_w$. We can control the output size by changing the kernel size. The output will be a *summary* of the input data with a lower height and width.

2.3 Speech recognition

Automatic speech recognition (ASR) is the problem of getting a program to automatically translate the sound of spoken language. In a traditional ASR (see Figure 2.7) the extracted audio features are inserted in the *acoustic model*, this model maps the relationship between the audio signal and a sequence of linguistic units.

Often this linguistic unit is in the form of a *phone* (called *phoneme* in linguistics), which distinguishes one word, or word element from another. The English language contains 44 phonemes. The *pronunciation model* captures the mapping between these phone sequences to words. We illustrate this function with an example of the word `apple` in figure 2.8.

With the help of a *language model*, we know the most likely ordering of words given the previous words (we will explain the details of a language model and how to create one in the next section). In general the traditional

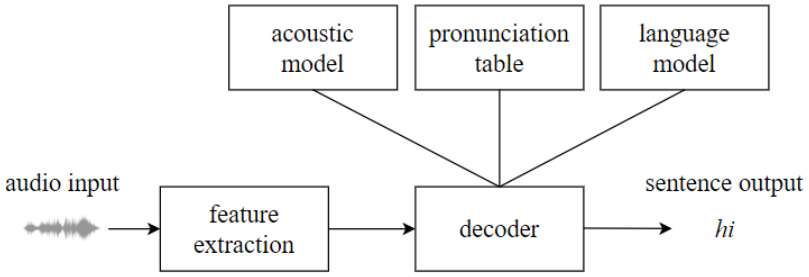

Figure 2.7: Traditional ASR architecture

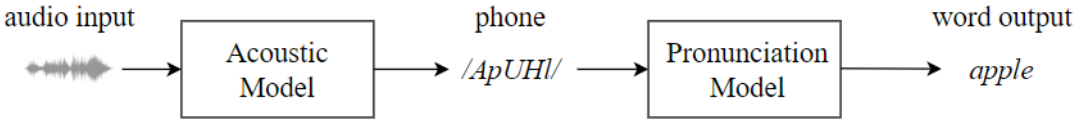

Figure 2.8: Acoustic & Pronunciation model

ASR tries to model:

$$\hat{P}(audio) = \hat{P}(audio|transcript) * \hat{P}(transcript)$$

Where we take the arg max of the possible transcripts given from our language model. One aspect of traditional ASRs is that the three models are often trained separately using data with different input and output labels. Traditional ASRs introduce a lot of complexity by continuously translating data to different forms eventually leading to the transcript output. The *end to end* architecture (Figure 2.11) simply takes audio input and outputs a transcript, all components within the model are trained together trying to achieve the same goal. Training is combined with *Connectionist temporal classification* [9] loss. It is common to use a language model [25] to improve our word sequence predictions. To compute the output sequence S we take the *arg max* of the probability of the sequence according to the language model $P_{lm}(S)$ multiplied with the *language model weight* lw.

$$S = \arg\max(P_{lm}(S) * lw)$$

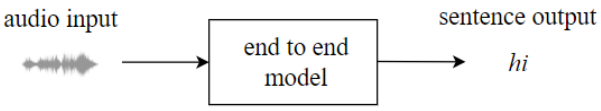

Figure 2.9: End to end model architecture

2.3.1 Language models

A language model [25] tries to capture the most likely orderings of words generated. The key concept for a language model is capturing the *n-grams* where $n \in 1, .., N$ and N is the largest N we want to capture. In a language model, we try to capture all the possible n-grams. An n-gram is a probability for a sequence of n words to occur. We try to calculate for k words:

$$P(x_1, x_2, ..., x_k) = \prod_{i=1}^{k} P(x_i | x_1, ...x_{i-1})$$

But the latter term is too complex to be estimated, that is why we make an approximation limited by n:

$$\hat{P}(x_1, x_2, ..., x_k) = \prod_{i=1}^{k} P(x_i | x_{i-n+1}, ...x_{i-1})$$

For example when we try to calculate the probability of the 2-gram `Good morning`, our calculation should be:

$$P(\text{Good}, \text{morning}) = P(\text{Good})P(\text{morning}|\text{Good})$$

When creating a language model we try to estimate these probabilities using a *corpus*. We can capture the probability of `Good` like this:

$$\hat{P}(\text{Good}) = \frac{n(\text{Good})}{k}$$

Where $n(word)$ is the number of occurrences of *word*. We can do something similar to get for the probability of `morning` given `Good`:

$$\hat{P}(\text{morning}|\text{Good}) = \frac{n(\text{Good}, \text{morning})}{n(\text{Good})}$$

2.3.2 Word error rate

The *word error rate* (WER) is a metric that is commonly used to measure the effectiveness of an ASR model by comparing the output sentence (hypothesis) with the actual transcription of the audio (reference). We can

compute the WER as:

$$\text{WER} = \frac{S + D + I}{N}$$

Where S is the number of substitutions, D the deletions, I the insertions and N the number of words in the reference sentence. The WER is a number between 0 and 1, the closer the WER is to 0 the larger the similarity between the hypothesis and the reference sentence.

2.3.3 Connectionist temporal classification

The process of *feature extraction* in traditional ASRs is trying to chop the audio file into small pieces for each timeframe and figuring out which *phone* was said during this timeframe. This process is difficult because we do not know exactly where the sound of a phoneme starts and ends. Let's take the example of the sound `hi` during 4 timeframes $t_0 \rightarrow t_3$:

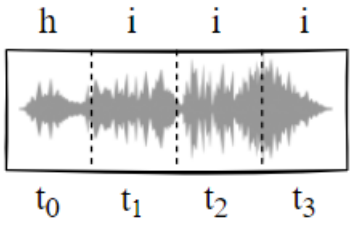

Figure 2.10: Audio file segmentation

When extracting the features of this audio file we get duplicates for the *phone* `i` as illustrated in Figure 2.10. *Connectionist temporal classification* (CTC) is used in end to end speech recognition, it solves this issue by introducing the blank `-`. The blank is a pseudo-character added during encoding and will be removed during decoding. It is added when there is a *transition* between two phones, it also helps encoding sequences with two identical subsequent output tokens (like the sequence `hello`). `h-ii` will become `hi` by applying the *CTC trick*:

- Remove double characters
- Remove blanks

The encoding of the label `hi` using CTC is illustrated in Figure 2.11.

When calculating the loss, we consider all possible *paths* for the word in our label (for example `hi`). Some possible paths to get to this word are `h-i-`,

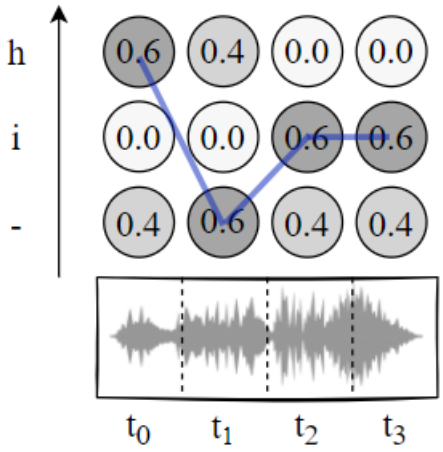

Figure 2.11: CTC: weights and paths

`h-ii`, `h--i`, `-h-i`, etc. We sum the scores of these paths and the result, the closer this number is to 1 to smaller our loss. When decoding an audio file, we perform the same process, and for each timeframe, we take the phone (or blank) with the largest weight and apply the *CTC trick*. For example, applying the CTC trick to `aaa-p-p-l-e` will give us the label `apple`.

2.3.4 Wav2vec2

Wav2vec2 [6] training happens in two phases: *pre-training* and *fine-tuning*. During pre-training, the model performs self-supervised learning where the model tries to predict *masked* speech units from the audio, similar to BERT. A difference with BERT is that the speech audio is not segmented into words or other speech units, but units of 25ms (shorter than phones) which enables high-level contextualized learning for many aspects of the audio. The fact that the set is finite means that the model can only focus on a subset of the aspects of audio, which encourages the model to ignore aspects like background noise. In contrast, other self-supervised approaches encourage the model to reconstruct the entire audio signal, where background noise is also learned.

The architecture of Wav2vec2 is presented in Figure 2.12. First the *feature encoder* consisting of multiple CNN blocks learns the speech representations:

$$f : X \longrightarrow Z$$

The *quantization module* takes these representations Z and turns them into learning *targets* Q or *codebooks*, codebooks, in simple terms they are summaries of the linguistics at certain timestamps. The transformer block uses

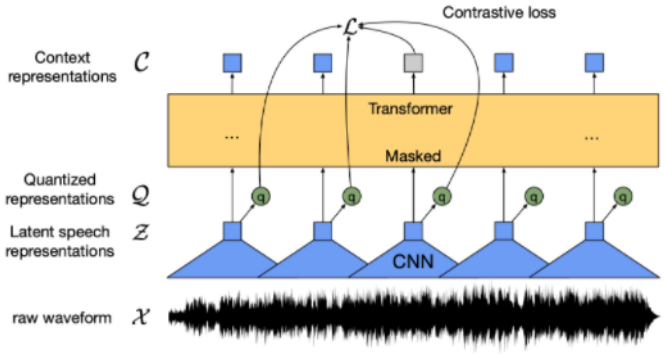

Figure 2.12: Wav2vec2 architecture [6]

CNN layers to add information to its input, part of this input Z is masked at certain timestamps:

$$g : masked(Z) \longrightarrow C$$

To compute the loss the model tries to identify the true quantized representations Q from *distractors* for the masked positions. Where the distractors K are other quantized representations.

During *fine-tuning*, we are using labeled data (much lower amounts of data compared to the unlabeled pre-training data) where the model tries to minimize the CTC loss.

Fairseq [2] provides several models pre-trained on different *LibriSpeech* [3] speech datasets (unlabeled). For example, the XLSR model is pre-trained on 128 languages with approximately 436K hours of unlabeled speech data.

Chapter 3

Fine-tuning Wav2vec2

To compare the performance of Wav2vec2 trained on audio data with regular text labels and closed captions we have to train a model with closed captions. In this research we will perform different experiments on Wav2vec2:

- We will test the effectiveness of Wav2vec2 when fine-tuning with labels in the form of captions. We will edit the design of the experiments to find out what will improve the effectiveness.
- We will investigate the properties of captions that Wav2vec2 can capture.
- We explore the pre-processing steps that would be required to get promising results.
- We try to find out what the effect of using different amounts of data in terms of hours of audio data.

3.1 Dataset

In this research we will be working with audio data from NPO broadcasts. This audio data was recorded by SpraakLab [4] and saved as WAV files with varying lengths between 1 to 60 minutes. The audio data was recorded in the period between 2021-05-01 to 2021-06-05. Later in the research we also collected audio data from the period 2022-01-01 to 2022-01-03. The audio files are broadcasts from channel 1 including various types of programs such as commercials, live broadcasts, news shows and exercise programs. The captions corresponding to these audio files are created by NPO and recorded once again by SpraakLab [4].

When training the Wav2vec2 models for this research we will be using the data from the period 2021-05-01 to 2021-05-31 as our *training* dataset. The *validation* dataset, which will be used to compute intermediate test results

during training, is drawn from the period 2021-06-01 to 2021-06-05. After training, we will manually compute our own test results from the model using an *evaluation* dataset, which includes the data from the period 2022-01-01 to 2022-01-03. During the evaluation process of this research, it became clear that it was essential our *evaluation* dataset needed to originate from a period far away (in terms of time) from our *training* dataset. If the *evaluation* dataset was adjacent or too close to the *training* dataset it could happen that the *evaluation* dataset included reruns of a program originally broadcasted during the *training* dataset period (for example reruns of news broadcasts).

3.1.1 File formats

The audio files are recorded as `WAV` files, compared to for example `MP3` files `WAV` files are uncompressed which allows the audio to be more consistent with the recording without losing any of the noise of the recording (noise is also important because it allows the model to learn to ignore it). To illustrate `MP3` is limited to a frequency of 44.1 kHz (44.1k audio samples per second) compared to 96 kHz for `WAV` files. The captions are delivered as `VTT` files which include the start- and endtime (accurate to 3 decimal seconds) of a caption and its text, as illustrated in the example:

```
00:25:52.404 --> 00:25:55.878
Iedereen wil weten wat de jurk wordt, maar het wordt geen jurk.

00:25:55.878 --> 00:25:57.997
Mogen we hem zien? Wil je hem zien?

00:25:57.997 --> 00:26:00.041
Ik ben sowieso fan van Diana Ross.
```

The input to Wav2vec2 will be a series of audio-label pairs. Each caption label will be a single caption accomponied with the corresponding audio, in other words the audio starting at *starttime* and ending at *endtime*.

3.1.2 ASR data

In addition to the audio files and caption data, we will also be using ASR transcriptions corresponding to the audio files. We fine-tuned an XLSR Wav2vec2 model on 300 hours of labeled Dutch audio, originating from the Dutch Speech Corpus [19]. An important note here is that the labels of this data are no captions, the speech corpus was transcribed in verbatim text. Next, using this trained model we generated labels for our audio files. The labels are lists of elements containing information of what word was said at

what timestamp in a .hyp file. We give an example of one of these elements:

```
{"wordID": "Dit", "beginTime": 4.36, "endTime": 4.5, "confidence": 0.99}
```

The function of this ASR data is that it helps us to get an idea of what was actually said by the speaker during a time window, this is different than the caption during this same time window. This data will not be used for training Wav2Vec2. By knowing the difference between the caption and what was said we can measure the quality of the caption to a certain degree. Another function of this data is that it will serve as a *baseline* model to compare the models with that we train on caption labels.

We summarized the data that we will be using for this research in Figure 3.1.

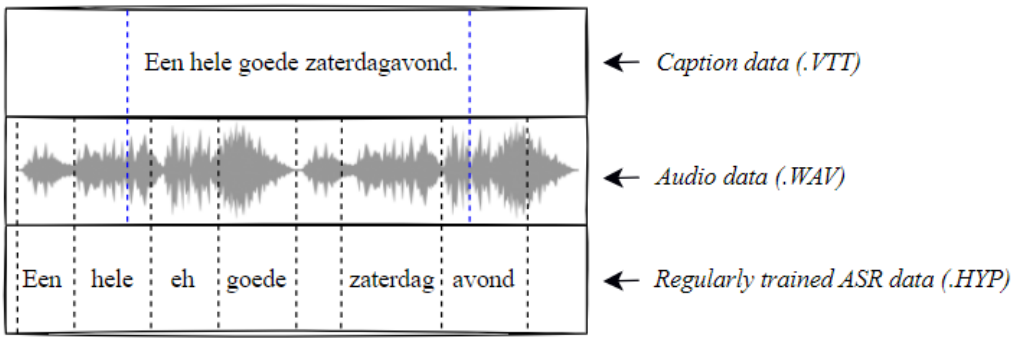

Figure 3.1: Research data

3.1.3 Data quality

The most important aspect of the data used in this research is the quality of the captions. The process of creating captions is done by TT888 [5] interpreters, who follow guidelines when creating the caption text and deciding the begin- and endtime when a caption is shown on screen. As explained earlier we are using data from different types of captions. A valid question to ask is whether the process of creating these captions is different for different broadcasts. However, the interpreters are following the same guidelines when interpreting captions for different programs. Therefore we can consider the quality of the data mostly homogeneous concerning different types of broadcasts. Interpreters are not limited by time restrictions when creating captions for these programs. This means that they have time to go back and fix certain captions when they noticed they made a mistake. This does not apply for live broadcasts, interpreters receive live broadcasts 1

minute earlier to create captions that will be broadcast. This is of course not enough to create flawless captions because interpreters are also under a lot of pressure. Interpreting broadcasts is done by one person at a time. They work for 30 minutes at a time when creating captions for live broadcasts to reduce errors.

3.1.4 Caption guidelines

Interpreting captions is a difficult task, it is important that all interpreters follow the same guidelines to make the captions as homogeneous as possible. The main goal of interpreting is to create captions where the essence is put forward. People who watch with captions need to get a good idea of the essence but also get an idea of how the speaker presents the information. Interpreters try to improve the readability, while not omitting too many words, all while also not exceeding the *word-per-minute* (wpm) rate of 180. Interpreters at NPO use software to check the wpm rate during the interpreting process. We list the most important aspects of the guidelines that are relevant in this research:

- *Direction instructions:* The captions can include phrases that are written in *all caps* to give instructions about the audio. Examples of this are **MET LIMBURGS ACCENT, MUZIEK, LIVE-UITZENDING, MENSEN PRATEN DOOR ELKAAR**.
- *Capital letters:* All sentences should start with a capital letter
- *Interpunction:* Usage of interpunction (`:` `,` `.` `?` `!` `...`) should be done if it improves the readability of a sentence. ... is used when the sentence does not fit on the screen, the next subtitle will start without a capital letter to indicate the continuation of the sentence.
- *Fillers:* Usage of fillers such as `eh`, `ehm`, `m-hm`, `hahaha` is allowed if it is important to understand what is happening on the screen. But they are mostly omited, often because they can be distractive to readers but also to comply with the 180 wpm rule.
- *Language mistakes:* Always fix language mistakes, unless it is relevant for the watcher. An example where a language mistake could be relevant would be at a comedy show. *Numbers:* Numbers from 0 to 10 are written in full. Numbers larger than 10 are written in their numeral form.

3.2 Filtering data

To fine-tune wav2vec2 properly we need high-quality data, meaning data that will help Wav2vec2 best to learn how to recognize captions. We need

to apply a filter to our data to omit low-quality or irrelevant data. First, we filter on entire broadcasts and then we filter on individual utterances within broadcasts. Some of the entire broadcasts are not useful, we apply *broadcast-filters* to make sure we never use this data during training. We apply *caption-filters* to filter out individual utterances within a broadcast, we filter on individual captions since some of the captions within a broadcast might still be useful for training. Since the amount of data we have is limited we will enable/disable some of these filters for training in chapter 4. Another reason that we do this is to see the effect of these filters on the quality of the data.

3.2.1 Broadcast-filters

We use the following filters to omit entire broadcasts from training:

- *Duplicate data:* After analyzing the data, a big part of it ($\pm$40%) turned out to be duplicate. Even though it matters little for the quality of the fine-tuning if duplicate data exists, the training process does go a lot faster with less duplicated data. We also want to avoid having the same data in the training and test set.

- *Live broadcasts:* This data is omitted completely from the training data. As mentioned earlier the quality of the captions for live broadcasts is lower. Omitting this data will make our data more homogeneous.

- *Commercials:* We also completely omit this data. Mostly because they contain a lot of web links, which can be difficult to learn in speech recognition. They also contain a lot of music and other sounds, which is not the focus of this research.

3.2.2 Caption-filters

We use the following filters on individual captions:

- *Unsynchronized captions:* After analyzing the data closely we found out that some of the caption texts accompanied start- and endtimes that were not synchronized with the corresponding audio. Obviously, this data should not be used for training. To find out which captions were unsynchronized we compared the *baseline ASR labels* with the reference caption labels during the same time window. We quantified the similarity of these labels by computing the WER by taking the *baseline ASR labels* as the hypothesis and the `VTT` captions as reference. If the WER was above the threshold of 50% we consider the caption *unsynchronized* and omit it for training.

- *Regular expressions:* Some words that occur in the caption are difficult to learn. For example words that contain numbers like `1e`, `2e`, `10.000` or web links, etc. The following regular expression formats are omitted:
  - `.{1,}-.{1,}`
  - `[A-Z]{1,}`
  - `[0-9]{1,}.[0-9]{1,}`
  - `[0-9]{1,}e`

3.3 Pre-processing data

Before training, we will perform some pre-processing on the data. A big part of the pre-processing already took place during filtering, where we omitted parts of the data. In this section, we will discuss methods that we used to *edit* the usable data. Editing the data is done for the following reasons:

- *Comparing learned properties:* To find out what Wav2Vec2 can learn and what it can not learn. By comparing the quality of models that were trained on datasets with different *configurations* (each configuration allows for the use of certain characters in the captions labels) we can speculate about the properties of the captions that the model learned.
- *To improve training:* We do not expect our model to perfectly learn all properties of the captions. By removing or editing parts of the caption labels we make sure that Wav2Vec2 will not have to focus on learning these properties.

3.3.1 Editing captions

We perform the following edits on the caption labels that are used for training:

- *Numbers to words:* We translate all numbers to words that do not contain numeral values. For example, `51` will be translated to `eenenvijftig`. Teaching Wav2vec2 how to recognize numbers is something we decided not to focus on within the scope of this research.
- *Editing individual characters:* We perform the following edits on certain characters within the caption labels:
  - é $\rightarrow e$
  - è $\rightarrow e$
  - ë $\rightarrow e$

- ê → e
- ö → o
- ó → o
- ï → i
- ü → u

3.3.2 Character configurations

We will produce different datasets, all the datasets will contain the same audio data and caption labels. The difference between the datasets is the characters that occur within the caption labels. Each configuration will be produced by *omitting* a subset of characters from the caption labels and *allowing* other characters to remain:

- *Configuration 1:* 28 characters including {`[a-z] ' |`}. (`|` is the *word-separator*, which Wav2Vec2 uses to indicate a `space` between words.) This configuration is very minimal and only allows for lowercase alphabet characters, it also includes the `'` symbol.
- *Configuration 2:* 34 characters including {`[a-z] ' | . , ? ! - :`}. This configuration adds *interpunction* to the captions, and aspect which we hope Wav2Vec2 can learn.
- *Configuration 3:* 63 characters including {`[a-z] [A-Z] ' | . , ? ! - : % & ‘`}. This configuration mainly adds *capital letters*. It also adds several rare characters..

When creating a *dictionary* as explained in section 3.4.1 each configuration will have a different dictionary.

3.3.3 Shifting start- & endtimes

In figure 3.1 we can see that the caption label is not always synchronized exactly with the corresponding audio fragment. The start- and endtimes are not in the same timeframe where the speaker starts and stops talking. In the figure we can see that the *baseline ASR data* shows us that words are being said outside of the time window where the caption is shown. The are several reasons for this:

- *Guidelines:* Interpreters have to conform to guidelines, for example staying within a wpm rate of 180, which sometimes forces them to use different start- and endtimes than the spoken audio fragment.
- *Human errors:* When interpreting spoken audio the interpreters could make errors causing imperfect start- and endtimes.

- *Recording limitations:* When recording the captions merely the start-times were recorded. The starttime of the caption of index $i+1$ was used as the endtime of the caption with index i. Except when the difference between the starttime of the caption $i+1$ and the starttime of the caption i was more than 3 seconds, in this case, the endtime of the caption i would be the starttime plus 3 seconds.

Within this research, we will investigate whether we can use these imperfect start- and endtimes in our training data to still get a good performance. To research this properly we will also train a model with 'improved' start- and endtimes, in other words, start and end- times that do correspond to the spoken audio more accurately.

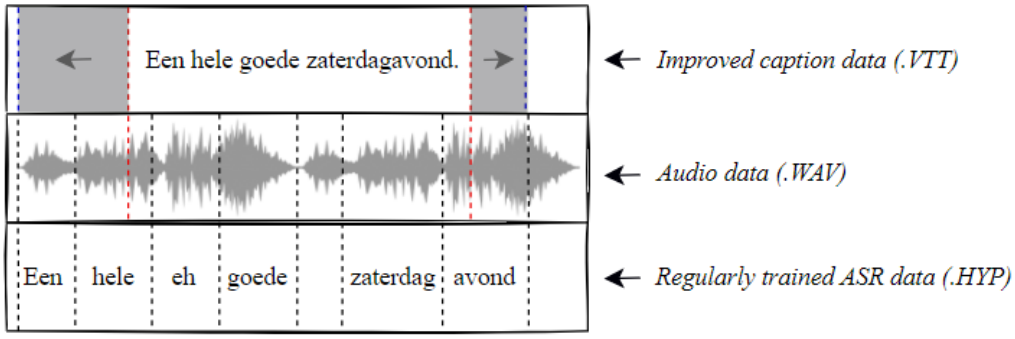

Figure 3.2: Improved caption data

The technique we use to generate these improved caption labels is shown in Figure 3.2. We take the starttime T and subtract a value a from it and then calculate the WER with the *baseline ASR data*:

$$T_{new} = T - a$$

Now calculating the WER with this T_{new} will give us a different WER value. We will repeat this process for different values of a, within the range $[-1.0, 2.0]$. We will use the T_{new} value where the WER compared to the *baseline ASR data* was lowest.
We will repeat this process for the *endtime* where we try different values of *add_time* within the range $[-1.0, 2.0]$.

3.4 Training

To train the model we will only focus on *fine-tuning* with labeled captions. We will not be doing any *pre-training* on the model since this process requires

a lot of unlabeled audio data, often months of it, which takes a lot of time to obtain in high-quality. In addition, the pre-training process itself takes weeks on efficient and fast systems. Because of time constraints, this is simply not feasible. But more importantly, it is not necessary, *Fairseq* already provides many pre-trained models trained on high-quality audio data. The goal of pre-training is to let the model learn how speech and language are structured regardless of how this is translated to text. Therefore providing different audio data to improve the learning process of captions was not necessary, the aspect of learning where it tries to translate audio to text happens only during fine-tuning of the model.

We will fine-tune Wav2vec2 using *Fairseq*. Fairseq [2] is a sequence modeling toolkit that allows researchers and developers to train custom models for translation, summarization, language modeling and other text generation tasks. For this research we will be using two pre-trained models:

- *BASE10:* This pre-trained model is trained on 960 hours of the *Librispeech* dataset [3]. The configurations we use in addition to this model, which was optimized for approximately 10 hours of labeled audio data, can be found in Appendix 7.1.
- *XLSR:* This model is pre-trained on 436K hours of MLS, CommonVoice and BABEL [1] data. This data originates from 53 languages, including Dutch. The configurations we use in addition to this model can be found in Appendix 7.2.

3.4.1 Dictionary file

To fine-tune a model using *Fairseq*, the software requires a *dictionary* file. This dictionary file is generated using all caption labels used for training. The dictionary is a list of all characters occurring in the caption labels and their *frequencies*. Fairseq uses this information to create new caption labels using the characters in the dictionary.

3.5 Model evaluation

To evaluate our model manually we will define an *evaluation* dataset. As explained earlier the audio data was recorded in the period between 2021-05-01 to 2021-05-31. Additional files of the period 2021-06-02 to 2021-06-05 are also included. These additional files for the month of June is what we will be using as our *evaluation* dataset.

We can use *Fairseq* to get the *hypothesis* of the label of an audio file that a Wav2vec2 model resulted from finetuning came up with. To help fairseq

with this process, we have the option to use *language models* which helps to create sentences given a finite set of word combinations and probability distribution. The language models used to evaluate our fine-tuned models are trained with all the caption labels that were available to us, except for the validation and evaluation set labels (including the captions we omitted during *filtering*) using KenLM. KenLM [11] is a language model inference program we can use for training a language model. The caption labels used for our language models will be pre-processed the same way our training data is pre-processed as described in section 3.3.

Apart from *language models* we also have the option to change certain parameters used by *Fairseq* when creating hypothesis sentences, these parameters include:

- *LM weight:* This parameter will influence the use of our language model. A very large *LM weight* or lw will practically disable our ASR model and predict words merely based on the language model. The default value we use for our experiments is $LMweight = 2$.
- *Word score:* Apart from the LM_weight, the $wordscore$ or ws will also influence the final hypothesis output sentence S. In short when the $wordscore$ is very negative our model will lean towards using a small number of words w, and when it is very positive it will lean towards using a lot of words. The default value for this is $wordscore = -1$.

The model tries to predict an output sentence S using the following formula, where P_a is the Wav2vec2 acoustic model:

$$S = \arg\max(P_a(S) + lw * P_{lm}(S) + w(S) * ws)$$

To evaluate the effectiveness of a fine-tuned Wav2vec2 model we will compute the WER between the *hypotheses* of the model on the audio data W2V2_{hyp} in the *validation* dataset and the caption labels in the *validation* dataset caption labels *cl* (our reference):

$$\text{WER}_{\text{W2V2}} = \text{WER}(\text{W2V2}_{\text{hyp}}, \text{cl})$$

As a *baseline* to compare this WER to we will also compute the WER between the *hypotheses* of the *baseline ASR model* on the audio data ASR_{hyp} in the same *validation* dataset and the caption labels in the *validation* dataset *cl*:

$$\text{WER}_{\text{ASR}} = \text{WER}(\text{ASR}_{\text{hyp}}, \text{cl})$$

Chapter 4

Experiments and results

In this chapter, we will present the different designs of our experiments and their results. The motivation behind the experiments was often based on results and analyses of their former experiments. Thus we will present most of the experiments in chronological order.

For each experiment, we will train three Wav2Vec2 models, each training will contain different amounts of data. To collect more data we alleviated some of the filters:

Dataset	*Live-broadcast filter*	*Commercial filter*
8 hours	✓	✓
16 hours	✗	✗
32 hours	✗	✗

When using both the *Live-broadcast filter* and *Commercial filter* on our *training* data we were able to collect no more than 8 hours of audio data. When disabling the *Commercial filter* we would only end up with approximately 8.5 hours of audio data. Which would not make for a very interesting dataset, since it was so similar to the 8-hour dataset. So we decided to disable the *Live-broadcast filter*, which gave us about 32 hours of audio data. the 8-hour dataset is 0% live broadcasts and commercials, the 16-hour dataset 50% and the 32-hour dataset 25%.

4.1 Fine-tuning BASE10 with different character configurations

In section 3.3.2 we discussed the use of different *character configurations*. We will investigate the differences in effectiveness for these configurations using the BASE10 model variant of Wav2vec2. We discussed that for each

configuration our caption labels cl will contain different characters. In this experiment we train three BASE10 models on configurations c:

$$c \in config_1, config_2, config_2$$

We compute the WERs for the systems s BASE10 and our *baseline* ASR:

$$\mathrm{WER}(\mathrm{hyp}_{(\mathrm{BASE10,c})}, \mathrm{cl_c})$$

$$\mathrm{WER}(\mathrm{hyp}_{(\mathrm{ASR})}, \mathrm{cl_c})$$

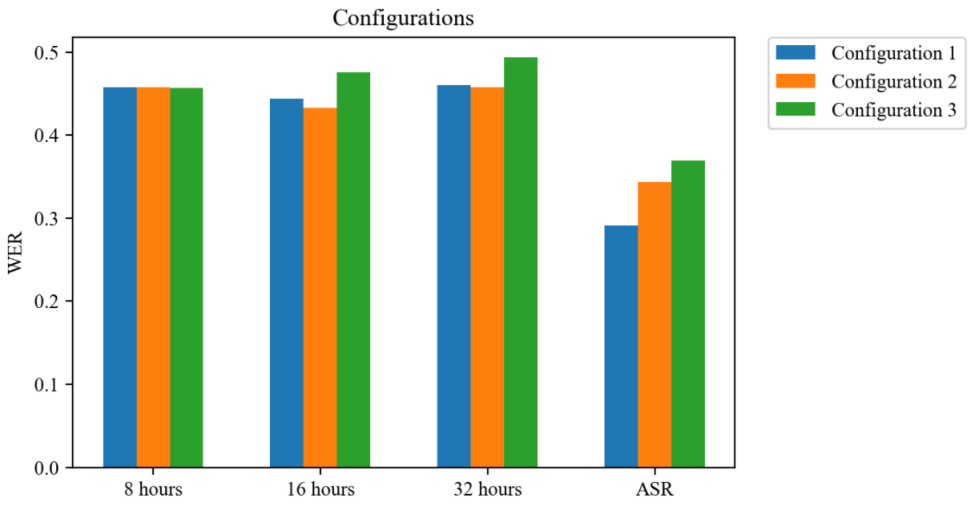

Figure 4.1: Experiment 1

These results tell us that BASE10 performs worse on captions than our *baseline* ASR when transcribing captions regardless of the character configuration. Our models trained on different amounts of audio data show no clear difference in performance. So clearly using more audio data will not have any effect for this model. Configuration 2 shows the best results. In addition, the *baseline* ASR model performs best on $config_1$, worse on $config_2$ and worst on $config_3$. This difference exists because the ASR model, in contrast to BASE10, was only trained on labels including characters of $config_1$. So the model performs worse as we introduce more characters in the character labels cl

4.1.1 Training details

When fine-tuning BASE10 using *Fairseq* we used the configuration shown in Appendix 7.1. For each configuration, we trained for 20k iterations. On

average this process took roughly 8 hours and 30 minutes, we found no notable differences between the configurations or amount of labeled audio data in terms of training time. All of the configurations converged to a minimum WER after roughly 18k iterations, after this there was little to no improvement. In other words, the *loss* did not decrease much further.

4.2 Fine-tuning XLSR

In this experiment, we will be training the XLSR variant of Wav2vec2 as explained in section 3.4. In our last experiment, we found that *configuration* 2 yields the best results. In the remaining experiments, we will only be training this configuration. We compare the following WERs:

$$\text{WER}(\text{hyp}_{(\text{BASE10,config}_2)}, \text{cl}_{\text{config}_2})$$

$$\text{WER}(\text{hyp}_{(\text{XLSR,config}_2)}, \text{cl}_{\text{config}_2})$$

$$\text{WER}(\text{hyp}_{(\text{ASR})}, \text{cl}_{\text{config}_2})$$

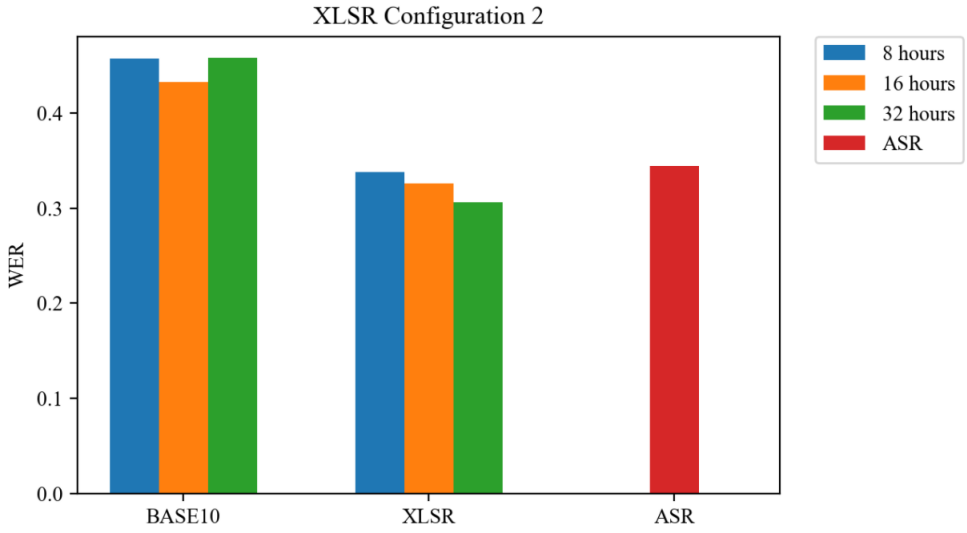

Figure 4.2: Experiment 2

In figure 4.2 we can see that the XLSR variant performs much better than BASE10. In addition, the XLSR variant performs better than our *baseline* ASR on all 3 datasets (8, 16 and 32 hours). Our XLSR model also seems to improve its performance when more hours of audio data are used, this shows that we could use more than 32 hours of audio data to improve the

performance further. BASE10 is pre-trained exclusively on English audio, but XLSR is pre-trained in many languages including Dutch. This factor clearly contributes significantly to the performance of the model.

4.2.1 Training details

When fine-tuning XLSR using *Fairseq* we used the configuration shown in Appendix 7.2. We trained for 60k iterations. On average this process took roughly 2 days and 17 hours. We found no notable differences between the amount of labeled audio data in terms of training time. Interestingly, after roughly 15k iteration the *training loss* increases while our *validation loss* decreases. There seems to be taking place some overfitting during training. When we train with more audio data there is less overfitting. After roughly 15k iterations our 8 and 16-hour models do not improve anymore, while our 32-hour model does improve slightly throughout roughly 40k iterations.

4.3 Tuning evaluation parameters

As discussed in section 3.5, there are default values for the *LM weight* and *word score* parameters when inferring *hypotheses* for a Wav2vec2 model. The default values (also our baseline) for these parameters are $LMweight = 2$ and $wordscore = -1$. In this experiment, we investigate the effect of small changes to these parameters. Note that we did not train any additional Wav2Vec2 models for this experiment. We only tuned parameters when inferring *hypotheses* labels. We performed this experiment on our three variants of $(\text{BASE10}, \text{config}_2)$ and $(\text{XLSR}, \text{config}_2)$ with different lengths of audio data.

4.3.1 Tuning LM weight

First, we tune the $LMweight$ parameter on the range $[0, 4]$ with the step size $= 0.1$, where we use the default value $wordscore = -1$. For our BASE10 variant in figure 4.3 if we look at the point where our $LMweights$ are 0 (no usage of the language model) our BASE10 model seems to perform better when we fine-tune with more labeled audio data. We collected the best $LMweight$ for each model variant:

- *8 hours:* $LMweight = 2.0$
- *16 hours:* $LMweight = 1.2$
- *32 hours:* $LMweight = 1.2$

Clearly when our model is trained on fewer hours of labeled audio data we require a larger dependence on the language model to perform optimally.

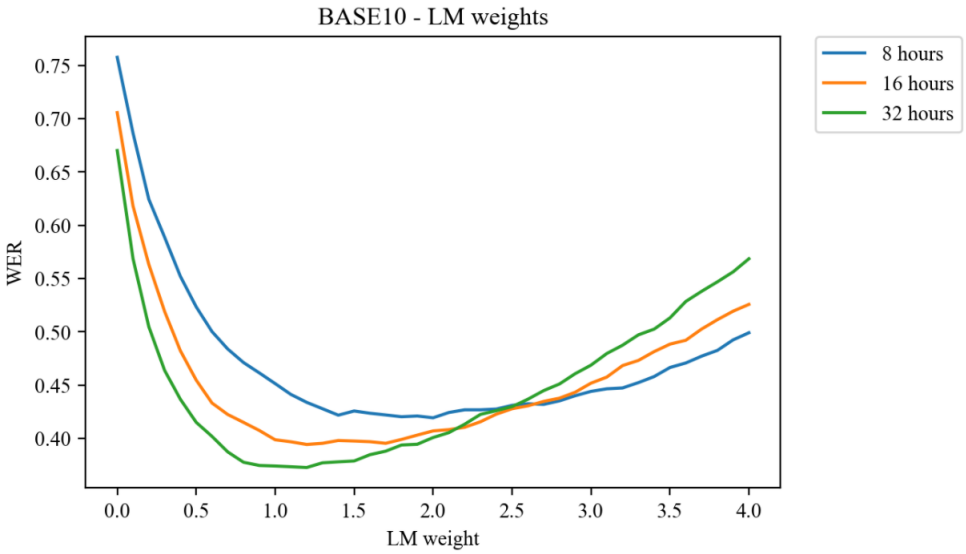

Figure 4.3: Experiment 3

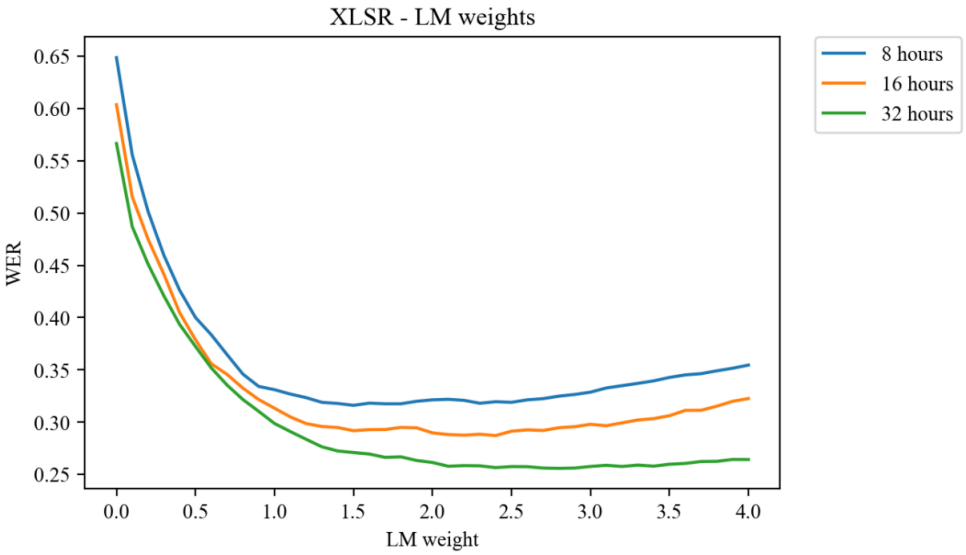

Figure 4.4: Experiment 4

Interestingly, as the results of the XLSR variant, shown in Figure 4.4, XLSR relies much more on the language model than BASE10. Here the best performing values for $LMweight$ are:

- *8 hours:* $LMweight = 1.5$
- *16 hours:* $LMweight = 2.4$
- *32 hours:* $LMweight = 2.8$

In contrast to BASE10 the XLSR model relies more on the language model when the amount of labeled audio data increases.

In the subsequent experiments, we will be using these $LMweight$ values for the evaluation of our models.

4.3.2 Tuning word score

Next, we tune the *wordscore* on the range $[-2, 2]$ with step size $= 0.1$, where we use the default value $LMweight$ of 2:

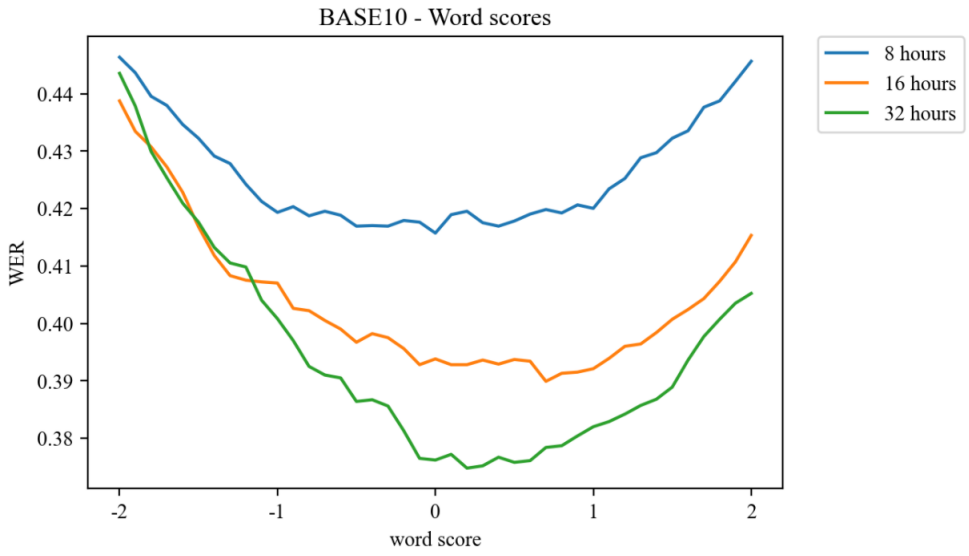

Figure 4.5: Experiment 5

The best WER for BASE10 in figure 4.5 are the following:

- *8 hours:* $wordscore = 0.0$
- *16 hours:* $wordscore = 0.7$
- *32 hours:* $wordscore = 0.2$

The results in Figure 4.5 contain a lot of noise, but roughly the best performing *wordscore* seems to be around 0.0. In other words, our models perform best on captions when there is no additional penalty or reward at all on the length of the output sentence. It seems that Wav2vec2 already learned by itself how many words are needed for the best performance when transcribing captions.

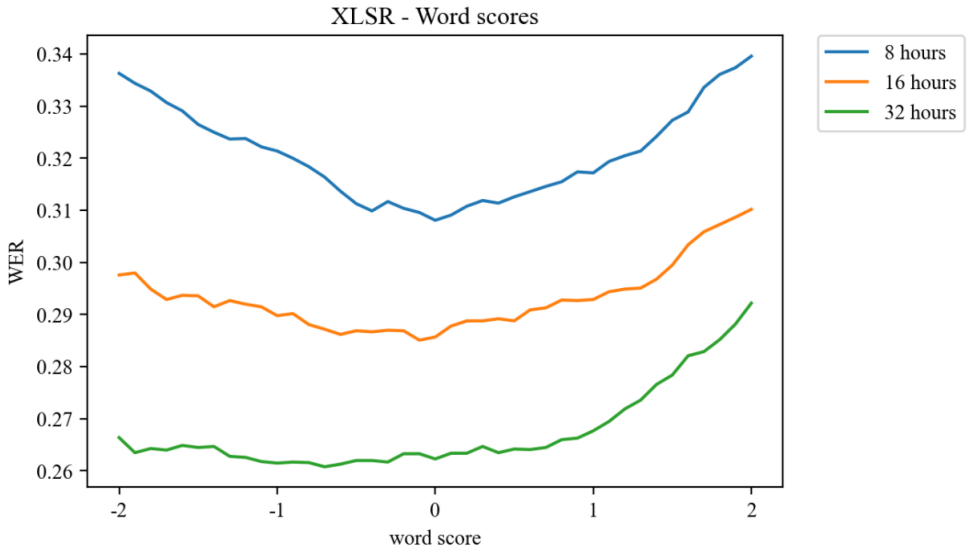

Figure 4.6: Experiment 6

For the XLSR variant in Figure 4.6 the results are similar to BASE10:

- *8 hours:* $wordscore = 0.0$
- *16 hours:* $wordscore = -0.1$
- *32 hours:* $wordscore = -0.7$

Once again the optimally performing *wordscore* seems to be around 0.0. Additionally, when comparing Figure 4.5 and Figure 4.5 we see that the performance of BASE10 is more dependent on the choice of the *wordscore* than our XLSR model is.

In the subsequent experiments, we will use the parameter value 0.0 for *wordscore*.

4.4 Shifting caption start- & endtimes

For our next experiment, we will be experimenting with *shifting* the start- and endtimes of captions, as described in section 3.3.3. We will take our caption labels with characters from configuration 2 ($\text{cl}_{\text{config2}}$) and edit the start- and endtimes to get shifted labels ($\text{cl}_{\text{config2,shifted}}$). These labels will be used to train our models. We compare this to a model trained on $\text{cl}_{\text{config2}}$. To get a good idea of how well this works we will perform this experiment on both the BASE10 and XLSR variants of the model. We compare these results with the evaluation of the *baseline* ASR on $\text{cl}_{\text{config2,shifted}}$.

4.4.1 BASE10

For the first experiment on BASE10, we will compare the following WERs, where BASE10-S is BASE10 trained on shifted captions.

$$\text{WER}(\text{hyp}_{(\text{BASE10,config}_2)}, \text{cl}_{\text{config2,shifted}})$$

$$\text{WER}(\text{hyp}_{(\text{BASE10-S,config}_2)}, \text{cl}_{\text{config2,shifted}})$$

$$\text{WER}(\text{hyp}_{(\text{ASR})}, \text{cl}_{\text{config2,shifted}})$$

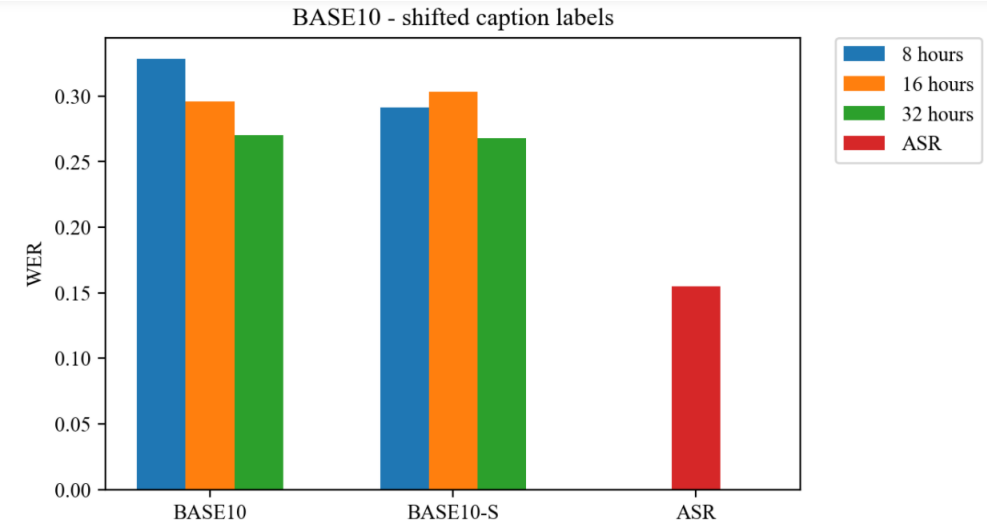

Figure 4.7: Experiment 7

If we compare Figure 4.2 and Figure 4.7 we can see that both our BASE10 model and the *baseline* ASR get much better performance on the $\text{cl}_{\text{config2,shifted}}$ captions. The $\text{cl}_{\text{config2,shifted}}$ are *synchronized* much better with the corresponding audio. This makes speech recognition easier because the audio

corresponds better to what was actually said by the speaker. We discuss this phenomenon in more detail in section 4.7. In addition, we get slightly better performance than BASE10 with our BASE10-S model because it was also trained on the higher quality (more synchronization and less noise) $\text{cl}_{\text{config2,shifted}}$ labels. Clearly shifting the captions using our *baseline* ASR model makes it easier for the BASE10 to recognize captions.

4.4.2 Training details

When fine-tuning BASE10-S using *Fairseq* we used the configuration shown in Appendix 7.1. We trained for 20k iterations. On average this process took roughly 8 hours and 30 minutes, once again we found no notable differences between the amount of labeled audio data in terms of training time. All of the configurations converged to a minimum WER after roughly 18k iterations, after this there was little to no improvement.

4.4.3 XLSR

Next, we will compare the following models, where XLSR-S is XLSR trained on *shifted* caption labels.

$$\text{WER}(\text{hyp}_{(\text{XLSR,config}_2)}, \text{cl}_{\text{config2,shifted}})$$

$$\text{WER}(\text{hyp}_{(\text{XLSR}-\text{S,config}_2)}, \text{cl}_{\text{config2,shifted}})$$

$$\text{WER}(\text{hyp}_{(\text{ASR})}, \text{cl}_{\text{config2,shifted}})$$

The results, shown in Figure 4.8, are similar to BASE10, we get a small improvement when training our XLSR model on shifted caption labels. Interestingly, for both our BASE10-S and XLSR-S model we get worse results on our models trained on 16 hours of audio data. Yet both the 8 and 32-hour models perform better than BASE10 and XLSR.

4.4.4 Training details

When fine-tuning XLSR-S using *Fairseq* we used the configuration shown in Appendix 7.2. We trained for 60k iterations. On average this process took roughly 2 days and 17 hours. We found no notable differences between the amount of labeled audio data in terms of training time. Similar to our evaluation of XLSR in section 4.2.1 we find some overfitting, for XLSR-S the difference between *training loss* and *validation loss* does seem to be smaller. There seems to be taking place slightly less overfitting. Similar to XLSR we notice that our 8 and 16-hour models do not improve after roughly 17k iterations, where our 32-hour model slowly improves over 40k iterations.

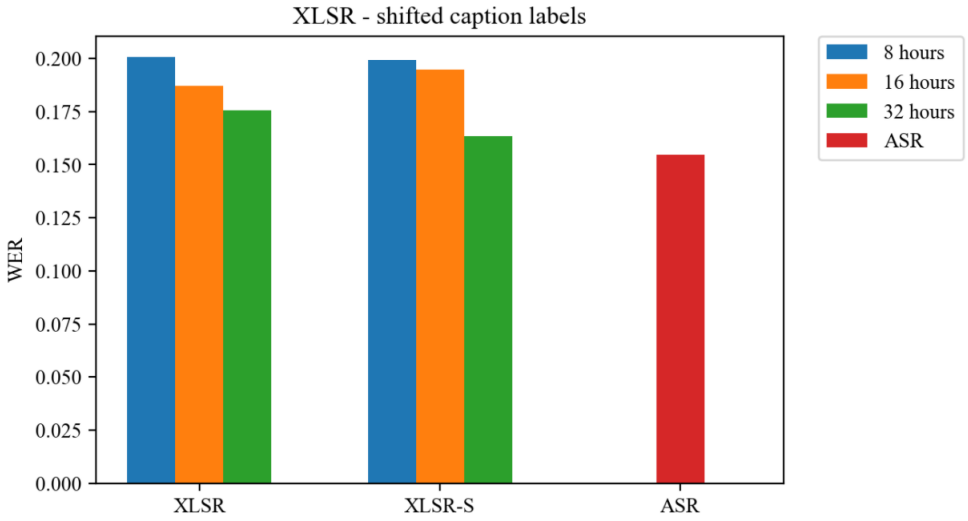

Figure 4.8: Experiment 8

4.5 Same language models

In our last experiments, we used different language models to create hypothesis labels when evaluating our models. As explained in section 3.5 for the models that we trained ourselves we used the caption labels from our dataset to train the language models. For our *baseline* ASR model, we used a 3-gram language model trained on 5 paper Dutch annuals. These generated labels have an error rate of ±10% and the characters of the words are limited to 53 characters. These include the alphabet `[a-z]`, the alphabet in uppercase `[A-Z]` and a `space` character.

In this experiment, we evaluate our best model XLSR-S trained on 32 hours of labeled audio data with shifted caption labels of configuration 2 ($\text{cl}_{\text{config2,shifted}}$) using the same language model that was used to evaluate the *baseline ASR model*. However, this language model does not contain *interpunction*, which was characteristic for configuration 2. For this experiment, we consider two language models:

- *No interpunction:* This will be exactly the language model explained above. (What we used for our *baseline* ASR model)
- *Punctuation:* The *no interpunction* language model does not contain probabilities for interpunction, but configuration 2 does have interpunction. We trained a unigram model using the *dictionary* file (section 3.4.1) corresponding to configuration 2. This unigram model that

includes interpunction was mixed together with our *no interpunction* language model using *SRILM* [23] where our unigram model has a weight of 10%.

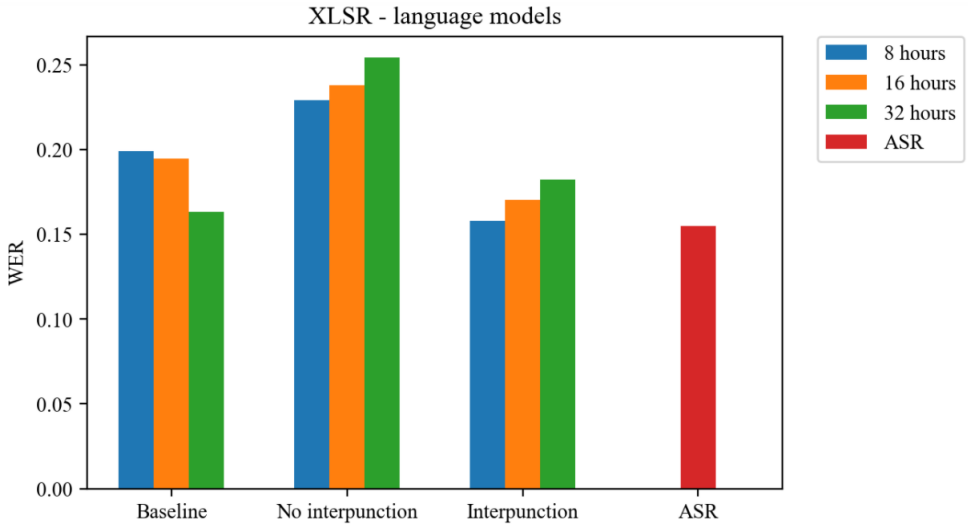

Figure 4.9: Experiment 9

When we look at the results in Figure 4.9 the first observation we make is that our *No interpunction* language performs the worst. This makes sense since the language model does not contain interpunction, while the captions in configuration 2 do. Furthermore, the model trained on 8 hours performs better on the *Interpunction* language model than our baseline. Lastly, both our *Interpunction* and *No interpunction* model perform increasingly worse on our models trained with more audio data (16 and 32). We suspect that this might have to do with the evaluation parameters: *wordscore* and $LMweight$. These parameter values are optimized for our baseline language model. Optimizing these values for the different language models might result in even better performance

4.6 Capturing properties of captions

In this section, we will explore details of the output caption labels given by XLSR-S trained on 32 hours of labeled audio data. Which is the model we fine-tuned in section 4.4.3, our best performing model. The *evaluation* dataset on which we evaluated this model contains 310 caption labels. We try to get a better understanding of the properties of captions that the model

Symbol	Evaluation set occ.	Hypotheses occ.	Correct usage
:	4	0	0%
,	19	14	42.9%
.	393	278	93.5%
?	22	3	100.0%
!	4	1	100.0%
...	49	20	63.6%

Table 4.1: Interpunction XLSR

Symbol	Evaluation set occ.	Hypotheses occ.	Correct usage
:	4	0	0%
,	19	11	45.5%
.	393	284	93.3%
?	22	3	66.7%
!	4	1	100.0%
...	49	19	72.7%

Table 4.2: Interpunction XLSR-S

was able to capture, and how it compares to our the output of our *baseline* ASR.

4.6.1 Interpunction

In this section we explore the use of interpunction symbols (: , . ? ! ...). For each symbol we count the number of occurrences of the symbol in our *evaluation* set, the number of occurrences in our output hypotheses caption labels and what percentage of these occurrences were correct plus in the right position of the sentence (correct usage). We performed this experiment on both our XLSR and XLSR-S models. The results are presented in table 4.1 and 4.2

The results are very different for all of the interpunction symbols. The . symbol seems to be easiest to learn for Wav2vec2, this makes sense because the majority of the caption labels end with a . which makes it easy to learn both the position and the usage. For the , we get a correct usage of around 45% which is surprisingly high. The ? symbol was also learned quite well, but was a lot less common in our hypothesis labels. The ! symbol

was not very common in our dataset, but Wav2vec2 still picked up on its existence. Wav2vec2 was not able to learn the `:` symbols. `:` is very rare in our dataset which makes it a lot more difficult for Wav2vec2 to learn. The `...` depends on the subsequent caption sentence (as explained in section 3.1.4). Yet Wav2vec2 still learned the symbol quite well, it is possible that Wav2Vec2 learned that when sentences are very long and the audio fragment is cut off mid-sentence a `...` symbol would be appropriate. The `...` is also the only symbol where we have a significant difference between the correct usage of our XLSR and XLSR-S models. XLSR-S performs much better on this symbol, possibly because the mid-sentence cut-off is more apparent.

4.6.2 Fillers

Now we investigate the use of fillers including `eh`, `uh` and `euh` As explained in section 3.1.4 these words are omitted from the captions in most cases. We looked for occurrences of these fillers in the output labels of the *baseline* ASR and compared them with our model output labels. From the 49 occurrences of fillers in the output labels of the *baseline* ASR 44 of the output labels from the XLSR-S model contained NO fillers (90%). This clearly indicates that Wav2Vec2 almost perfectly learned to omit fillers. Below we illustrate this with an example we found in our results about polish supermarkets:

- *Original caption:*

  ```
  u weet vast dat eind vorig en begin dit jaar er een reeks van
  aanslagen was op poolse supermarkten.
  ```

- *XLSR-S hypothesis:*

  ```
  die weet vast dat eind vorig jaar begint dit jaar er een
  hele reeks van aanslagen was op polsen supermarkt.
  ```

- *Baseline ASR hypothesis:*

  ```
  u weet vast dat uh eind vorig jaar begin dit jaar haar een
  hele reeks van aanslagen was op Poolse supermarkten onder
  ```

4.6.3 Summarizing

Next, we explore the output sentence lengths to investigate how well our model *summarizes* the information in the audiofile. The sentences in our *evaluation* set have an average length of 13012/310 = 42.0 characters. In

contrast the hypotheses sentences of our XLSR-S model have an average length of 12516/310 = 40.4 characters. And the labels of our *baseline* ASR model have an average length of 13219/310 = 42.6 characters.

Our XLSR-S model seems to be the most conservative with words and summarizing too much. This is influenced by the *wordscore* parameter, which we experimented with in section 4.3.2. Increasing the *wordscore* would make our average sentence length closer to the true caption labels from our *evaluation* set. But this would be at the cost of increasing the WERs of our model.

The difference of average sentence lengths between our *baseline* ASR output and the true *evaluation* set caption labels shows us that the captions are indeed summarized slightly.

4.7 Caption quality & Limitations of baseline ASR

We discovered in section 4.4 that speech recognition became easier for Wav2Vec2 when we shifted the start- and endtimes of the caption labels to become more synchronized with the corresponding audio. We explained that we used the *baseline* ASR output labels to determine how synchronized the labels were. Secondly, in section 3.2 we explained how we used our *baseline* ASR to filter our *unsynchronized* labels. The reason we did this was that a large part of the training data was simply not usable for training. The audio fragments did not correspond in any way to the caption labels.

These two factors make the comparisons we made with our *baseline* ASR model a less meaningful because we used the hypothesis labels to pick out the labels for our *evaluation* set. In addition, we also edited the start- and endtimes of our caption labels in our *evaluation* set to be closer to the *baseline* ASR labels, which was followed by a comparison of our *evaluation* labels and our *baseline* ASR labels. This is why the results we collected between the models we trained ourselves and the *baseline* ASR model should be taken with a grain of salt. We showed that a *baseline* ASR is very useful to increase the performance of a Wav2Vec2 model trained on caption labels, but making a fair comparison to the same *baseline* ASR model became impossible.

Chapter 5

Related Work

In this chapter, we will explore work related to this research. The methods used in these studies did not have any impact on this study. The problem definitions in these studies are related to our research but not similar enough for the methods to be of any use.

5.1 Generating subtitles using automatic speech recognition

The focus of our research is investigating the use of caption data to train an ASR to generate captions. In [18] they train end to end ASR models on normal text labels (rather than caption labels). In this research they investigate different methods to create readable subtitles, where the only property of the subtitles is that they are summarized variants of the text that was spoken:

- Utilizing an unsupervised compression model to post-edit the transcribed speech and create readable subtitles.
- Modeling the length constraints within the end-to-end ASR system.

The second method turned out to achieve the best performance.

In [22] the focus is on creating software that takes a video and creates .SRT files (subtitles) as output. The speech recognition module used the research does not take caption properties such as 'summarizing' or adding interpunction into account.

5.2 Case studies on limits of Wav2vec2

In [26] it was stated that little research was done on Wav2vec2 in other languages besides English. They investigate the performance of Wav2vec2

in languages where little labeled audio data was available. It was found that Wav2vec2 learns basic acoustic units that can compose diverse languages. Also, Wav2vec2.0 can dynamically merge the fine-grained presentation into coarser-grained presentation to fit the target task.

Chapter 6

Conclusions

We have seen how Wav2vec2 performs using captions as labels rather than verbatim text. We learned that the data that we feed Wav2vec2 needs to be pre-processed carefully to create useful data for training. With the use of different filters and pre-processing techniques we learned what type of caption data is needed and how we need to edit it to improve the learning process. We also learned that the use of different types of models that Wav2vec2 provides, such as BASE10 and XLSR, can influence the learning process. We learned about several properties that Wav2vec2 can learn quite well about the captions.

We learned that working with caption labels is very difficult. From collecting the data to evaluating trained model results there are a lot of hassles that are being introduced which are not always visible early on.

6.1 Acknowledgements

We would like to thank David van Leeuwen for supervising this bachelor thesis. His experience with data science and Wav2vec2 made working in this difficult field of research a lot more manageable. We would also like to thank Kasper Brink for allowing us to work on the CN47 and CN99 cluster. Lastly, we would like to thank the human interpreters at NPO who took the time to tell us about the interpreting process of captions.

Bibliography

[1] Babel. https://babel.is.tue.mpg.de/.

[2] Fairseq. https://ai.facebook.com/tools/fairseq/.

[3] Librispeech. https://www.openslr.org/12/.

[4] SpraakLab - haalt meer uit audio en video. https://www.spraaklab.nl/.

[5] Tst888 ondertiteling. https://over.npo.nl/voor-publiek/toegankelijkheid/tt888-ondertiteling#Hoewerkt.

[6] Alexei Baevski, Henry Zhoum Abdelrahman and Mohamed Michael Auli. Wav2vec 2.0: A framework for self-supervised learning of speech representations. *Curran Associates, Inc.*, 33, 2020. https://arxiv.org/abs/2006.11477.

[7] Ludwig Boltzmann. *Studien über das Gleichgewicht der lebendigen Kraft zwischen bewegten materiellen Punkten.* k. und k. Hof- und Staatsdr, 1868.

[8] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou and Yoshua Bengio. A structured self-attentive sentence embedding. *Conference paper ICLR 2017*, 2017.

[9] Alex Graves, Santiago Fernández and Faustino Gomez. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. *In Proceedings of the International Conference on Machine Learning*, 60:369–376, 2006. https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.75.6306.

[10] Carrie Lou Garberoglio. Speech-to-text services: An introduction. pages 1–2, 2019. https://www.nationaldeafcenter.org/sites/default/files/Speech-to-Text%20Services_%20An%20Introduction.pdf.

[11] Kenneth Haefield. Kenlm. https://kheafield.com/code/kenlm/estimation/.

[12] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79, 1982. https://www.researchgate.net/publication/16246447_Neural_Networks_and_Physical_Systems_with_Emergent_Collective_Computational_Abilities.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser and Illia Polosukhin. Attention is all you need. *31st Conference on Neural Information Processing Systems (NIPS 2017)*, 2017. https://arxiv.org/abs/1706.03762.

[14] Yann LeCun. Object recognition with gradient-based learning. Springer, 1980. http://yann.lecun.com/exdb/publis/pdf/lecun-99.pdf.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of NAACL-HLT 2019*, page 4171–4186, 2019. https://arxiv.org/abs/1810.04805.

[16] Claude Lemaréchal. Cauchy and the gradient method. *Documenta Mathematica*, Extra Volume ISMP (2012) 251–254, 2012. https://www.math.uni-bielefeld.de/documenta/vol-ismp/40_lemarechal-claude.pdf.

[17] Aston Zhang, Zachary C. Lipton, Mu Li and Alexander J. Smola. *Dive into Deep Learning*. Amazon Science, 2019.

[18] Danni Liu, Jan Niehues and Gerasimos Spanakis. Adapting end-to-end speech recognition for readable subtitles. *Association for Computational Linguistics*, pages 247–256, 2020. https://arxiv.org/abs/2005.12143.

[19] Nelleke Oostdijk. Het corpus gesproken nederlands. http://neon.niederlandistik.fu-berlin.de/static/digitaal/digitaal-10.html.

[20] Herbert Robbins and Sutton Monro. A stochastic approximation method. *Ann. Math. Statist.*, 22, 1951. https://doi.org/10.1214%2Faoms%2F1177729586.

[21] Williams, Ronald J. Hinton, Geoffrey E. Rumelhart and David E. Learning representations by back-propagating errors. *Nature*, 323, 1986. https://ui.adsabs.harvard.edu/abs/1986Natur.323..533R/abstract.

[22] Abhinav Mathur, Tanya Saxena and Rajalakshmi Krishnamurthi. Generating subtitles automatically using audio extraction and speech recognition. 2015. https://ieeexplore.ieee.org/document/7078779.

[23] Andreas Stolcke. SRILM - an extensible language modeling toolkit. *Proc. Intl. Conf.*, 2002. http://www.speech.sri.com/projects/srilm/papers/icslp2002-srilm.pdf.

[24] Nik Vaessen and David A. van Leeuwen. Fine-tuning wav2vec2 for speaker recognition. 2021. https://arxiv.org/abs/2109.15053.

[25] Yoshua Bengio, Réjean Ducharme, Pascal Vincent and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2001. https://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf.

[26] Cheng Yi, Jianzhong Wang, Ning Cheng, Shiyu Zhou and Bo Xu. Applying wav2vec2.0 to speech recognition in various low-resource languages. *ArXiv*, abs/2012.12121, 2020. https://arxiv.org/abs/2012.12121.

Chapter 7

Appendix

```
common:
  fp16: true
  log_format: json
  log_interval: 200
  tensorboard_logdir: ./tensorboard

checkpoint:
  save_interval: 50
  save_interval_updates: 10000
  keep_interval_updates: 1
  no_epoch_checkpoints: true
  best_checkpoint_metric: wer

task:
  _name: audio_finetuning
  data: ???
  normalize: false
  labels: ltr

dataset:
  num_workers: 6
  max_tokens: 1600000
  skip_invalid_size_inputs_valid_test: true
  validate_after_updates: 10000
  validate_interval: 50
  valid_subset: valid

distributed_training:
  ddp_backend: legacy_ddp
  distributed_world_size: 1

criterion:
  _name: ctc
  zero_infinity: true

optimization:
  max_update: 20000
```

```
  lr: [0.00005]
  sentence_avg: true
  update_freq: [16]

optimizer:
  _name: adam
  adam_betas: (0.9,0.98)
  adam_eps: 1e-08

lr_scheduler:
  _name: tri_stage
  phase_ratio: [0.1, 0.4, 0.5]
  final_lr_scale: 0.05

model:
  _name: wav2vec_ctc
  w2v_path: ???
  apply_mask: true
  mask_prob: 0.65
  mask_channel_prob: 0.5
  mask_channel_length: 64
  layerdrop: 0.05
  activation_dropout: 0.1
  feature_grad_mult: 0.0
  freeze_finetune_updates: ???
```

Listing 7.1: base_10h.yaml

```
common:
  fp16: true
  log_format: json
  log_interval: 200
  tensorboard_logdir: ./tensorboard

checkpoint:
  save_interval: 1000
  save_interval_updates: 1000
  keep_interval_updates: 1
  no_epoch_checkpoints: true
  best_checkpoint_metric: wer

task:
  _name: audio_finetuning
  data: ???
  normalize: false
  labels: ltr

dataset:
  num_workers: 6
  max_tokens: 320000
  skip_invalid_size_inputs_valid_test: true
  validate_after_updates: 10000
  validate_interval_updates: 5000
  valid_subset: test

distributed_training:
  ddp_backend: legacy_ddp
  distributed_world_size: 1

criterion:
  _name: ctc
  zero_infinity: true

optimization:
  max_update: 60000
  lr: [0.0003]
  sentence_avg: true
  # 40
  update_freq: [40]

optimizer:
  _name: adam
  adam_betas: (0.9,0.98)
  adam_eps: 1e-08

lr_scheduler:
  _name: tri_stage
  phase_ratio: [0.1, 0.4, 0.5]
  final_lr_scale: 0.05

model:
  _name: wav2vec_ctc
```

```
w2v_path: ???
apply_mask: true
mask_prob: 0.75
mask_channel_prob: 0.25
mask_channel_length: 64
layerdrop: 0.1
activation_dropout: 0.1
feature_grad_mult: 0.0
freeze_finetune_updates: ???
checkpoint_activations: false
# to fix errors:
encoder_layers: 12
encoder_embed_dim: 768
encoder_ffn_embed_dim: 3072
encoder_attention_heads: 12
```

Listing 7.2: xlsr.yaml