Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Kristian Stensgård

Dynamic Beam Search Decoding for Speech Recognition using Confidence Estimation Module

Master's thesis in Electronic Systems Design and Innovation
Supervisor: Giampiero Salvi
Co-supervisor: Pablo Ortiz, Telenor Research
June 2022

NTNU
Norwegian University of
Science and Technology

Kristian Stensgård

Dynamic Beam Search Decoding for Speech Recognition using Confidence Estimation Module

Master’s thesis in Electronic Systems Design and Innovation
Supervisor: Giampiero Salvi
Co-supervisor: Pablo Ortiz, Telenor Research
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Abstract

Deep Learning has become increasingly popular over the last decade and has demonstrated promising results in computer vision and Natural Language Processing (NLP) tasks as well as for Automatic Speech Recognition (ASR). The use cases for this type of technology range from a convenience in our day-to-day life, with for instance the use of voice assistants, speech-to-text messaging and more. For the hearing impaired, this technology might have a bigger impact on their life and their experience in communication with other people. These systems are often implemented as purely data-driven systems and require a lot of data. In order to take benefit from all that data, models are also scaled up to have more parameters than ever before. Thus there is a need for better and more efficient models in order to make high-quality ASR systems accessible to everyone.

ASR systems tend to be overconfident in their predictions and strictly using the output probabilities is not a reliable method to quantify the confidence of the ASR. This is a problem for applications that require reliable and correct transcriptions, such as court hearings or medical recordings. The confidence scores are also used in voice assistants which will ask the user to repeat commands if the confidence for a command is low. If the confidence scores are reliable and accurate, the scores can be used for other use cases which involve better unsupervised training or efficient decoding. This thesis presents a few different methods to estimate the confidence of an ASR system. Then, a method for producing confidence scores for each token will be developed and used to estimate the confidence of an ASR system. This will be done by developing a Confidence Estimation Module, which uses the features from the ASR system input. This module has an AUROC of 0.83 and an AUC PR of 0.66, which translates into a decent confidence estimation module, but is not on par with the state-of-the-art. These confidence scores will later be used to do a dynamic Beam Search with the additional LM also developed in this thesis. The confidence scores from the CEM will be used to weigh the influence of the LM in the Beam Search. This is in an effort to reduce the number of errors in the transcriptions, with the added benefit of being easier to tune. The dynamic Beam Search shows signs of promising results under tuning but fails to outperform the standard static Beam Search, thus failing to meet expectations and not being consistent with the findings during tuning.

Sammendrag

Dyp læring med nevrale nettverk har blitt stadig mer populært det siste tiåret og har vist lovende resultater i datasyn og Natural Language Processing (NLP)-oppgaver samt for Automatic Speech Recognition (ASR). Bruksområdene for denne typen teknologi er mange og gjør hverdagen vår enklere, med for eksempel bruk av taleassistenter, tale-til-tekst for meldinger og mer. For hørselshemmede kan denne teknologien ha en større innvirkning på deres liv og deres erfaring med kommunikasjon med andre mennesker.

Disse systemene er ofte implementert som rene datadrevne systemer og krever mye data. For å dra nytte av all dataen, skaleres modellene også opp til å ha flere parametere enn før. Økningen i antall parametere fører til økte krav til hardware som bruke ASR systemene. Det er derfor behov for bedre og mer effektive modeller for å gjøre ASR-systemer av høy kvalitet tilgjengelig for alle.

ASR-systemer har en tendens til å være for sikker i prediksjonene sine, og det å bruke sannsynlighetsfordelingen som det nevrale nettverket produserer er ikke en pålitelig metode for å kvantifisere usikkerheten til en prediksjon fra ASR system. Dette er et problem for applikasjoner som krever pålitelige og korrekte transkripsjoner, dette kan være for eksempel rettsmøter eller medisinske opptak. Usikkerhets estimatet kan også brukes i stemmeassistenter som vil be brukeren om å gjenta kommandoer hvis usikkerheten for en kommando er høy. Hvis usikkerhets estimatene er pålitelige og nøyaktige, kan disse også brukes til andre oppgaver som bedre unsupervised-training eller effektiv dekoding.

Denne oppgaven presenterer noen forskjellige metoder for å estimere konfidensen til et ASR-system. Deretter vil en metode for å produsere en konfidensscore for hvert del ord bli utviklet og brukt til å estimere konfidensen til et ASR-system. En Confidence Estimation Module (CEM) vil bli utviklet til dette formålet. Denne modulen har en AUROC på 0.83 og en AUC PR på 0.66, dette tilsvarer en tilstrekkelig ytelse, men ikke på lik linje med state-of-the-art. Denne Confidence Estimation Modulen vil bli brukt i et dynamisk Beam Search sammen med ASR systemet og en språkmodell. Denne språkmodellen vil også bli utviklet i forbindelse med denne oppgaven. Dynamisk Beam Search er utviklet i håp om at det skal kunne redusere antall feil og være lettere å tune. Den dynamiske varianten av Beam Search viser lovende resultater under tuning for å finne hyperparameteren γ. Ved en full test som sammenligner dynamisk og statisk Beam Search derimot, viser det seg at den dynamiske varianten ikke fører til redusert feilrate. Dette er ikke i tråd med påstanden gjort tidligere som tilsier at den dynamiske varianten med en CEM gir bedre ytelse og er lettere å tune.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Background and motivation

In recent years large Natural Language Processing (NLP) models have become increasingly popular. Models like GPT-3 [1], T5 [2], and Switch-C [3], are NLP models that have received a lot of attention lately. These models showcase incredible performance and use cases span many industries and different areas. Tasks such as question answering, summation, translation, and sentiment analysis to name a few are what these models are capable of.

The models consist of billions, or even trillions, of parameters and, are trained using terabytes of data on large servers for several days [1]. Some of these models are open source and can be downloaded, hosted, and used by anyone, while some, like GPT-3, can only be accessed via a paid API. Even though pre-trained models are available to use by anyone, the fact that these models are mostly developed by large corporations like Meta, Google and Microsoft introduce a few challenges. These companies have a large influence over what data the models are trained on and the biases that may be introduced as a result. A huge reliance on these models can result in the companies having increased influence over the end user. Especially if the models are not open source and only accessible through an API, biases, censorship, and false information can easily be introduced to these models.

There are however some large open source models available to the public which enables smaller companies and individuals to integrate these models with their applications. Speech recognition is one of these applications, which is found in for instance voice assistants. A drawback with many common speech recognition systems, especially those that rely on large neural networks, is their inability to produce reliable confidence estimations for the transcriptions [4]. This feature is crucial in voice assistants, where the assistant will ask the user to repeat a command if the confidence for the command is low. Other use cases might include highlighting of words with low confidence in a transcription system. Thus the user does not have to proofread the entire transcription but only focus on low-confidence words. End users might not be directly the benefactors of a confidence score. This feature might help develop new models in a more efficient way. The confidence scores can be used when training a model in an unsupervised manner by efficiently using the data with appropriate confidence scores. For instance in a teacher-student setting, where a large pre-trained model, like Wav2Vec [5], is used to train a smaller model using unlabeled data. Using only data with at least a certain confidence will help the smaller student model to not learn erroneous text, and thus only use data which the teacher knows is correct.

1.2 Goals and research questions

This thesis will focus on confidence estimation techniques. More specifically how this can effectively be incorporated into an existing ASR model. Different methods for estimating the confidence of an ASR transducer model specifically, as well as more general approaches, will be introduced and their advantages and disadvantages will be presented. A system for producing a confidence estimate will also later be developed and trained using data from an ASR system. The thesis will provide insight into the question of if a confidence score for each word predicted by ASR system can reliably be produced.

Secondly, the usability of the confidence scores will be evaluated. This will be done by using the confidence scores for a dynamic Beam Search. In order to do this, a separate Language Model (LM) will be developed. The LM will add scores to the most likely next words in the transcribed sentence. The performance of the dynamic Beam Search will be evaluated against a normal static Beam Search.

1.3 Thesis Structure

This thesis will firstly cover how speech recognition and other speech-related tasks differ from vision and NLP tasks, due to the input data into a system. Then speech features and how features that are used in ASR are extracted from the raw waveform will be covered. Two different architectures commonly used for ASR will then be covered. How they are trained, advantages and disadvantages will be explained.

The thesis will then cover the foundation for the ASR system that is used in this thesis. Both the overall architecture and also how the individual components work.

A section on confidence estimation techniques will then be covered. Also, how these different techniques are implemented, trained, evaluated, and their corresponding advantages and disadvantages will be covered.

The implementation details of the models and training procedures for the models will be covered in the **Methodology and Experiments** section. The dataset and the experiments will also be covered in this section.

The following results and a discussion of these will be covered in the sections **Results** and **Discussion** respectively.

Lastly, a short summary and a conclusion of the experiments and their contribution to the initial questions raised in the **Goals and research questions** section will be presented.

This thesis is a continuation of the specialization project that took place fall of 2021, with the goal of developing an ASR system [6]. Some sections of the theory is derived from this project, with varying degree of modifications and extensions. Some are completely revised and others have only minor modifications.

2 Background Theory

2.1 Speech vs NLP vs Vision

As mentioned earlier, the NLP domain has been at the center of focus for new techniques and methods in the machine learning world. The introduction of the Transformer architecture and the multi-head attention mechanism in 2017 [7] can be seen as a paradigm shift in the machine learning domain. The transition from recurrent models, to feed-forward networks with attention resulted in groundbreaking results. The Transformer architecture and its derivatives have shown great performance on NLP tasks, which lead to a focus shift from vision tasks to NLP tasks. Vision tasks like object detection and segmentation were popular and great results on these tasks was attributed to convolutional networks. The convolutional networks are great for capturing local fine-grained features, which is hard with normal feed-forward layers. Later the Transformer architecture and the multi-head attention has also been adopted into vision tasks. The ViT [8] is one of the earliest successful attempts at this. However, the Transformer is not as effective as it is in NLP tasks.

The speech domain and Automatic Speech Recognition (ASR) task is a quite interesting and can be argued to lie somewhere in the middle between vision and NLP tasks. Speech is sequential just like text, however it requires feature extraction from a spectrogram which is normally done via convolutions, much like in vision tasks. Speech Recognition is a classification task, while a lot of NLP tasks are prediction tasks.

Speech Recognition may in theory benefit from an attention mechanism in a way that other words might be easier to classify based on the context of the sentence. Local context may also be more important for speech than for text. For instance, plosive sounds like “p” or “k” is likely followed by a vocal or an “r” and not another plosive.

In order to both capture local features and capture the sounds from a spectrogram, and also reason about words and their context, both convolutions, and multi-head attention are typically used in state-of-the-art ASR systems.

2.2 Speech Features

Speech is sequential in nature and is usually stored in data as a one-dimensional vector. Where the elements in the vector represent the magnitude of the signal. This representation is however not usually used as an input to a neural network. The raw waveform can be used for feature extraction, as in [5], although, traditionally transformation of the waveform is applied before being used in a neural network. A common representation is the spectrogram which shows the magnitude of the signal in various frequencies. Speech features can be extracted from the spectrogram using normal 2D convolutions, just like in vision tasks. The spectrogram is computed using the Fourier Transform over only a short time period, known as the Short-Time Fourier Transform (STFT). In order to emphasize the importance of the frequencies that are present in speech Mel-filter banks is also applied. The whole process of producing a spectrogram used in ASR can be broken down into four different steps:

- Framing
- Windowing
- Computing the power spectrogram
- Filtering with a Mel filter bank

The framing is essentially breaking up the signal into smaller and overlapping segments. A windowing function is then applied to each of these segments. For instance a Hamming window. These reduces the spectral leakage when the Fourier transform is applied to the segment. The Hamming window is expressed by Equation 1, with a_0 set to $\frac{25}{46}$. The window is of length N.

$$w[n] = a_0 + (1 - a_0)\cos\left(\frac{2\pi n}{N}\right), 0 \leq n \leq N \tag{1}$$

The windowed segments are then used to compute the power spectrogram by using the square of the absolute value of the Fast Fourier Transform (FFT).

$$P(x_i) = \frac{|FFT(x_i)|^2}{N} \tag{2}$$

In Equation 2, x_i is the entries in the segment $\mathbf{x}$ of length N.

The last step in the process is to use a set of band-pass filters. This set of filters, often referred to as an array of filters, are called a filter bank. These filters are evenly spaced on the Mel-scale and are symmetrical around their center frequency and have a magnitude of 1.

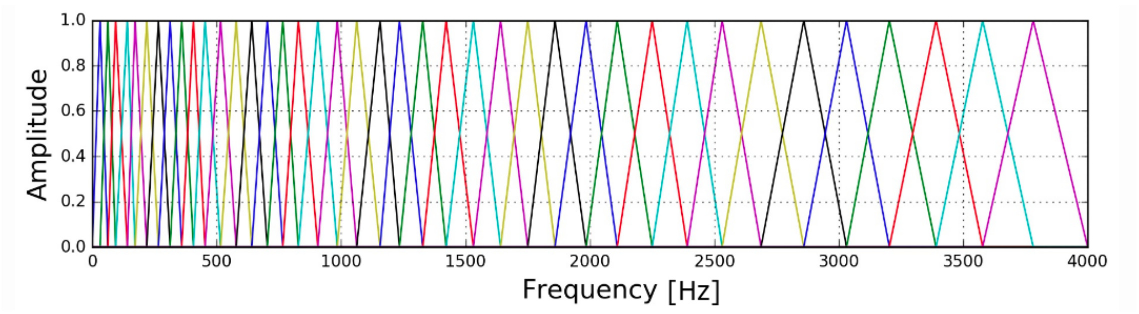


Figure 1: 40 filter banks on the Mel-scale [9]

Humans have a perception of speech, and sound in general, that is quite discriminative against low frequencies and less so against higher frequencies. Meaning that sounds with for instance $1kHz$ difference at the higher end of the frequency range would sound quite similar, but sounds with a $1kHz$ difference at the lower end of the frequency range sound very different. In other words, human perception of sound follow a logarithmic scale when it comes to frequencies. In order to mimic this effect, the power spectrogram is converted to Mel-scale which is a subjective scale, and therefore the values used in the conversion can seem a bit arbitrary [10]. The conversion between Mel, $[m]$ and frequency, f, is:

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right) \tag{3}$$

An example of a spectrogram used as an input in an ASR model is shown in Figure 2.

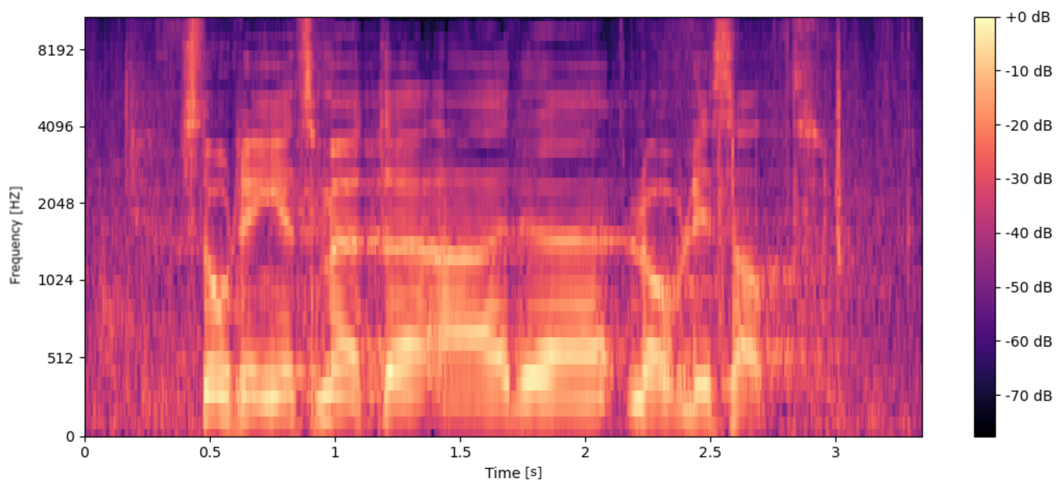


Figure 2: A spectrogram with 80 filter banks and an STFT length of 400, with windows of 10ms.

In order to increase the performance of a neural network, some alterations of the input data are often done between epochs. This alteration or augmentation can be the masking of words or tokens in the case of NLP tasks. For computer vision tasks this is done by introducing noise, blur, rotation, and other augmentations of the input image. Both of these techniques make it harder for the network to remember the input, and instead, the network has to learn the underlying features in the input. Thus the network will be less likely to overfit on the training data and perform better on the unseen test data. In speech recognition, augmentation can be done by applying some noise, filtering, speed perturbation, and masking, to name a few. A common method is to use SpecAugment [11]. As the name suggests, this is augmentations done to the spectrogram itself. The augmentation technique is a mix of masking and time warping. The masking is done on both the frequency and time axis. The masking along the time axis will mask a band of features across the whole frequency range in time. This will act as missing some parts of the spoken words. The masking along the frequency axis will mask a band of frequencies through the whole duration of the sample. This will prevent the model from relying too much on certain frequencies. The time warping will either speed up or slow down the speech sample, very much like a random speed perturbation, with the difference being that the perturbation is done directly on the spectrogram and not the waveform. An example of a spectrogram with this type of masking is shown in Figure 3.

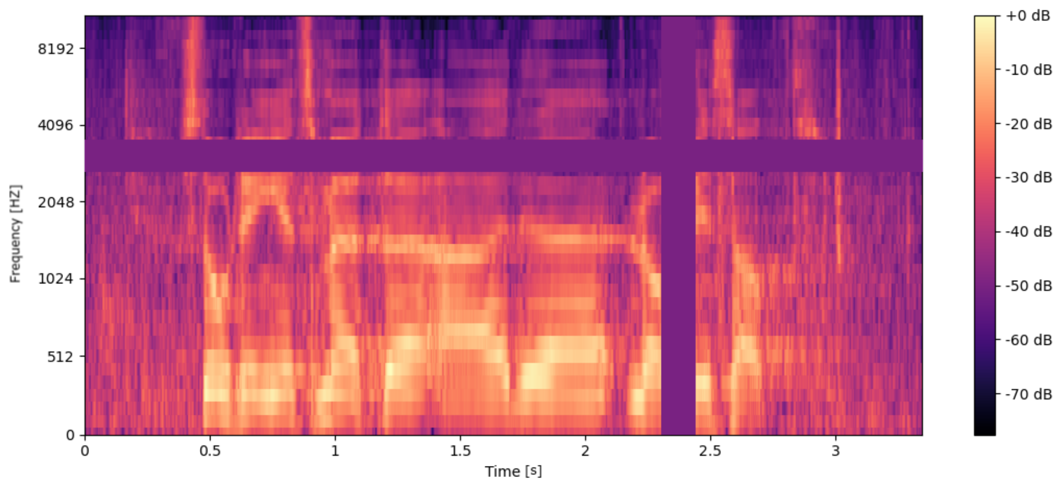


Figure 3: An example of SpecAugment with time and frequency masking.

2.3 Connectionist Temporal Classification

There exist a few different neural architectures for ASR, each with its own advantages and disadvantages. Probably the simplest of these architectures is a Connectionist Temporal Classification (CTC) model. This model uses only the acoustic features as input. The individual components in the model can be a wide variety of neural components, for instance, CNNs, RNNs, feed-forward layers, or attention components. The model is quite effective since it produces a single feature vector from acoustic input. This is in contrast to some other models which operate in an autoregressive way, which uses previous predictions as input and produces one token per iteration. The acoustic features are usually represented as a spectrogram with discrete time units. Speech is however not discrete in time, sounds can be uttered over several time units and start or stop in the middle of a time unit. The model thus needs to figure out the time alignment between characters and speech. In other words, it needs to be able to tell when in time a letter or sound is uttered and for how long this sound is uttered. This introduces another problem, however. It needs to be able to remove predictions of letters that correspond to the same letter or sound. For instance, if someone says "Car" over several time units it might produce a sequence like "CCAAARRRRRR", which is obviously wrong. This is solved by reducing repeating identical letters into a single letter. In order to deal with words that have repeating letters like the word "school", a blank token "-" is produced between the identical letters. A possible sequence for "school" might be "sschhoo-ool", which is then reduced to "school".

2.4 Transducer

The Transducer is another one of these architectures and one of the most widely used. Its strength is attributed to the fact that it has both an acoustic encoder and a token predictor. These are combined in a joint module which then produces a prediction based on both acoustic features and past predictions. This is in contrast to a single acoustic encoder used in a CTC model, which only uses acoustic features to produce a latent representation that is used in the prediction. The Transducer, essentially contains a Language Model (LM) represented by the predictor network, something the CTC lack. This means that the prediction network will "help" the acoustic encoder with the prediction, thus when the acoustic encoder is uncertain of a prediction, the prediction network will provide information about the most likely next token based on the previously predicted tokens. The encoder works the same way as the encoder in a CTC network, however, the latent representation is combined with a latent representation from a token prediction network. This combined representation of the acoustic prediction and the next token prediction is passed through a set of linear layers, and finally output as token probabilities.

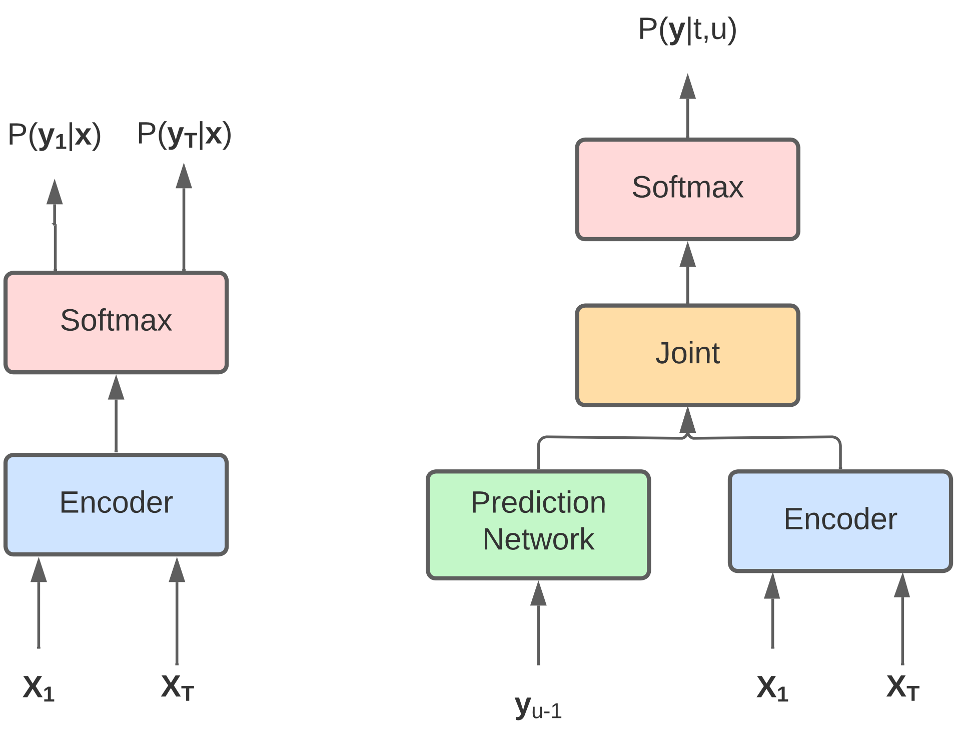


Figure 4: Illustration of two of the most popular ASR architectures. The CTC architecture on the left and the Transducer on the right.

In order to align features with labels, the Greedy Search algorithm can be used. This algorithm is relatively simple: predict the next token, and if the next token is not the blank token, append it to the output. The outputs $\mathbf{h_e}$ and $\mathbf{h_{t,u}}$ are only calculated if the respective variables t or u are updated in order to be efficient.

Algorithm 1 Greedy Search Algorithm

```
Require: $\mathbf{x} = \{x_1, \cdots x_T\}$ t:=1; u := 0; $\mathbf{y}$ := {}
 1: while $t \leq T$ do
 2:     $\mathbf{h_e} = f(\mathbf{x_t})$
 3:     $\mathbf{h_{t,u}} = g(\mathbf{y})$
 4:     $\mathbf{h} = h(\mathbf{h_{t,u}}, \mathbf{h_e})$
 5:     $p = \arg\max(\mathbf{h})$
 6:     if $p \neq \varnothing$ then                    ▷ If the prediction is a label
 7:         $u = u + 1$
 8:         add p to $\mathbf{y}$                         ▷ Add the token to the input
 9:     else if $p = \varnothing$ then
10:         $t = t + 1$
11:     end if
12: end while
```

The algorithm above, Algorithm 1, is called a greedy algorithm since it only uses the most likely output in the next forward step. Thus the most likely token is output, given the past predictions.

Another search algorithm is the Beam Search algorithm. It is called Beam Search since it keeps a beam of most probable predictions that are used as input to predict the next token. The algorithm keeps a set of these beams which it then updates at every time step t. The computational cost of beam search can be a lot higher because it needs to use each of the predictions in the beam as input to produce the next beam. Therefore it is common to use a greedy algorithm during validation and the Beam Search during testing or in production.

Algorithm 2 Beam Search Algorithm

```
 1: $B = \{\emptyset\}; P(\emptyset) = 1$
 2: $t \leftarrow 1$
 3: while $t \leq T$ do
 4:     $A = B$
 5:     $B = \{\}$
 6:     for $\mathbf{y}$ in $A$ do
 7:         $\mathrm{P}(\boldsymbol{y}) += \sum_{\hat{\boldsymbol{y}} \in \text{ pref } (\boldsymbol{y}) \cap A} \mathrm{P}(\hat{\boldsymbol{y}}) \, \mathrm{P}(\boldsymbol{y} \mid \hat{\boldsymbol{y}}, t)$
 8:     end for
 9:     while B contains less than W elements more probable
10:             than the most probable in A do
11:         $\mathbf{y}^*$ = most probable in A
12:         Remove $\mathbf{y}^*$ from A
13:         $\mathrm{P}(\mathbf{y}^*) = \mathrm{P}(\mathbf{y}^*) \, \mathrm{P}(\emptyset | \mathbf{y}, t)$
14:         Add $\mathbf{y}^*$ to B
15:         for $k \in Y$ do
16:             $\mathrm{P}(\mathbf{y}^* + k) = \mathrm{P}(\mathbf{y}^*) \, \mathrm{P}(k | \mathbf{y}^*, t)$
17:             Add $\mathbf{y}^* + k$ to A
18:         end for
19:     end while
20:     Remove all but the W most probable from B
21: end while
22: return $\mathbf{y}$ with the highest $\log \mathrm{P}(\mathbf{y}) / |\mathbf{y}|$ in B
```

In algorithm 2, the original Beam Search algorithm from [12] is shown. This Beam Search algorithm has a beam size of W, meaning that it will keep w of the most probable tokens and use them for the input in the next decoding step. $P(\mathbf{y})$ is the probability of emitting a sequence $\mathbf{y}$. The probability $P(k|\mathbf{y}, t)$ is the probability of extending the sequence $\mathbf{y}$ with k at time step t. k is a part of the set Y, which contains the most probable next tokens. pref($\mathbf{y}$) is the set of proper prefixes of $\mathbf{y}$. This means, that line 7 in algorithm is computing the total probability for the sequence, which is then length normalized at the last line of the algorithm.

Since the true alignment between the input sequence $\mathbf{x}$ and the ground truth sequence $\mathbf{y}$ doesn't exist, the transducer defines the probability $p(\mathbf{y}|\mathbf{x})$ as the sum of all possible alignments between $\mathbf{x}$ and $\mathbf{y}$ [13]. Since there may exist a vast amount of possible alignments, the forward and backward algorithm in [14] is used. The benefit of this method is that only a single forward pass of acoustic features and tokens before a backward pass is done and the gradients are calculated. In other words, no search algorithm needs to be used during training, only during inference. This greatly reduces the computational cost. The transducer model is trained at minimizing the loss function $\mathcal{L} = -\log p(\mathbf{y}|\mathbf{x})$. A disadvantage of this is the large memory footprint as the acoustic features and prediction features are joined into a single tensor. This new tensor has one dimension for the N acoustic time steps, one dimension for the M tokens, and one dimension d for the embedding dimension, which later is the significantly larger vocabulary dimension. This is ignoring the batch dimension.

2.5 Language model integration

A drawback with the Transducer model is that the prediction network is generally only trained on the transcription data. This amount of data is severely less than the amount of data that an LM that is trained to correct spelling and grammatical errors via masking. These types of models are trained on several gigabytes of data consisting of just text and have far more parameters and are therefore more precise and have a larger vocabulary [15]. This type of LM could be implemented to use the predictions from an ASR model and correct it or rescore them, reducing any grammatical or spelling errors. Two popular methods include; incorporating the LM in the Beam Search algorithm or doing N-best rescoring. When using an LM in the Beam Search decoding the probability predicted by the LM for the next token being k is added to the probability of it being k predicted by the transducer. In order to adjust the influence of the LM, a scalar weight is added to the probabilities for the LM. This method is also known as shallow fusion [16]. For N-best rescoring an LM is used to simply rank the N-best hypothesis from the beam search. This is done by choosing the highest scored hypothesis after weighting the scores from the LM with a weight γ and conversely the scores from the beam search with a weight of $1-\gamma$.

$$\mathrm{P}_{\text{rescored}}(y|x) = (1-\gamma)P_{BS}(y|x) + \gamma P_{LM}(y) \tag{4}$$

As mentioned earlier, the Transducer has outputs that are conditioned on the past outputs and not just the acoustic features like the CTC. This can introduce overconfidence in the output probabilities. The prediction network can overpower and win over the acoustic encoder when it is uncertain in its prediction. This will lead to the Transducer might outputting a sequence of tokens that is not at all what is in the speech sample, but a sequence that the Transducer has seen many times during training. In a Beam Search or a Greedy Search, the joint module is acting as a dynamic weighting module between the acoustic encoder and the prediction network, which is weighting the influence of the prediction network over the acoustic encoder. In a Beam Search with an external LM, this dynamic weighting is missing, the influence of the LM over the Transducer is fixed with a scalar weight. This means that the scalar weight for the LMs influence needs careful tuning not to overpower the Transducer.

2.6 Transformer

In 2017 a new type of architecture, mainly targeted at NLP tasks was introduced to replace RNN type of networks. The Transformer processes the inputs in parallel, instead of the sequential nature of an RNN type of network. The architecture has later been adapted to be used in vision and speech tasks, and nearly every other domain of machine learning. This type of architecture is referred to as a Transformer [7]. The full Transformer architecture is an encoder-decoder architecture that uses self-attention in the encoder and attention to another sequence in the decoder. Originally, the decoder would attend to the sequence output from the encoder. This means that the decoder would know how parts of the sequence from the encoder relate to the sequence in the decoder. The architecture consists of blocks of feed-forward modules, attention modules, and normalization modules. The overall structure of how this is put together is shown in Figure 5.

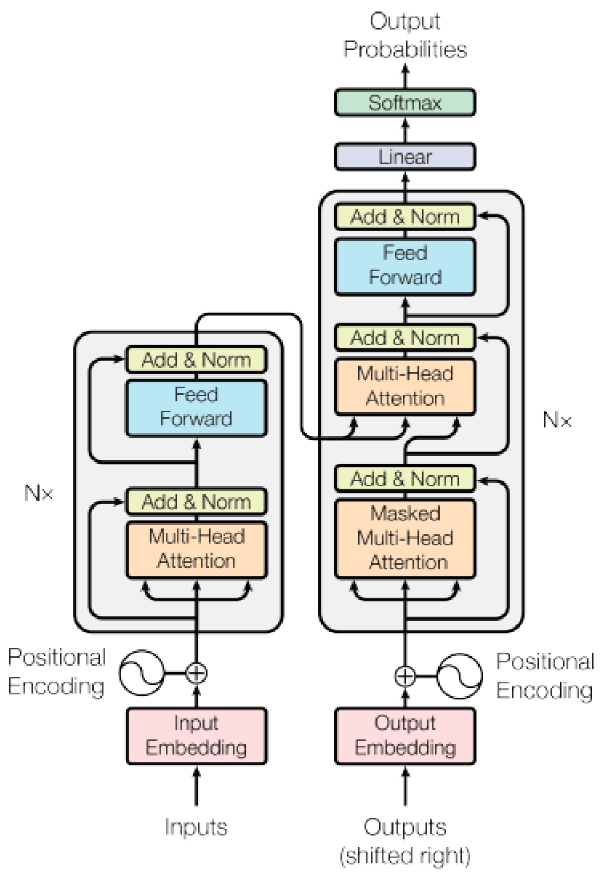


Figure 5: The original Transformer architecture, taken from [7]

The feed-forward layer consists of two linear layers and an intermediate activation function, σ, and can be expressed as:

$$\mathrm{FFN}(x) = \sigma(xW_1 + b_1)W_2 + b_2 \tag{5}$$

The feed-forward module has an expansion ratio of four, meaning that the output of the first and input to the second layer has a dimension of four times the size of the input x. The final output dimensions are the same as the input.

2.7 Attention mechanism

In recent years, the concept of an attention mechanism has gained popularity and has shown great results on a variety of tasks [7], [17], [18]. There exist different implementations of attention and attention mechanisms. For instance, attention can be location or content-aware. Also, there is a variety of methods combining different representations that are learned by the attention mechanism. In the following section, dot-product attention and the extension to multi-head attention will be discussed in further detail.

2.7.1 Dot product mechanism

The dot product attention was first introduced in [19] and also used in LAS [17]. This is equivalent to multiplicative attention. In the case of the LAS, two neural networks ϕ and ψ that both use a vector that is an audio feature vector and label feature vector respectively, that are produced by recurrent networks. The vector $\mathbf{e}$ is produced by taking the inner product of these, shown in Equation 6. A distribution is then created from the vector $\mathbf{e}$ by using the softmax operation, shown in Equation 28. Lastly, the context vector $\mathbf{c}$ is created by taking the element-wise multiplication of the two vectors α and $\mathbf{h}$, and summing the elements, shown in Equation 8. The resulting vector $\mathbf{c}$ consists of weighted features based on $\mathbf{h}$, see [17].

$$e_{i,u} = \langle \phi(\mathbf{s_i}), \psi(\mathbf{h_u}) \rangle \tag{6}$$

$$\alpha_{i,u} = \frac{\exp(e_{i,u})}{\Sigma_u \exp(e_{i,u})} \tag{7}$$

$$c_i = \Sigma_u \alpha_{i,u} \mathbf{h_u} \tag{8}$$

2.7.2 Multi-head attention

Another more recent attention mechanism is the multi-head attention mechanism, first introduced with the paper Attention is all you need [7]. The concept is more or less the same, by providing a way of weighing the feature vectors that are used in a network. The aforementioned paper also introduced the Transformer architecture. The multi-head attention mechanism can either be used in self attention or attention between different feature vectors. The “multi” in multi-head attention refers to multiple attention layers running in parallel. This allows the model to attend to different parts of the sequence, for instance, one head can be responsible for global context and another for local context. The attention layers consist of scaled dot-product attention layers, which differ slightly from the dot-product attention. The difference is that the vectors used to produce the vector $\mathbf{e}$ are scaled by a factor of $\sqrt{d_k}$ before using the softmax function, which results in the vector $\mathbf{e}$ having a variance of 1. The inputs to the scaled dot-product attention V, K and Q are referred to as the values, keys, and queries respectively. The value d_k is the dimension of the column space for the queries, Q, and the keys, K. The basics of the transformer is that it compares the ’query’ with the ’keys’ and gets weights for the ’values’. The mechanism is shown as a block diagram in Figure 6. On the left is the scaled dot-product attention and on the right is the complete multi-head attention mechanism. As seen in Figure 6, the multi-head attention produces an output vector that is a linear combination produced by multiple scaled dot-product attention operations in parallel. How this is done will be explained shortly.

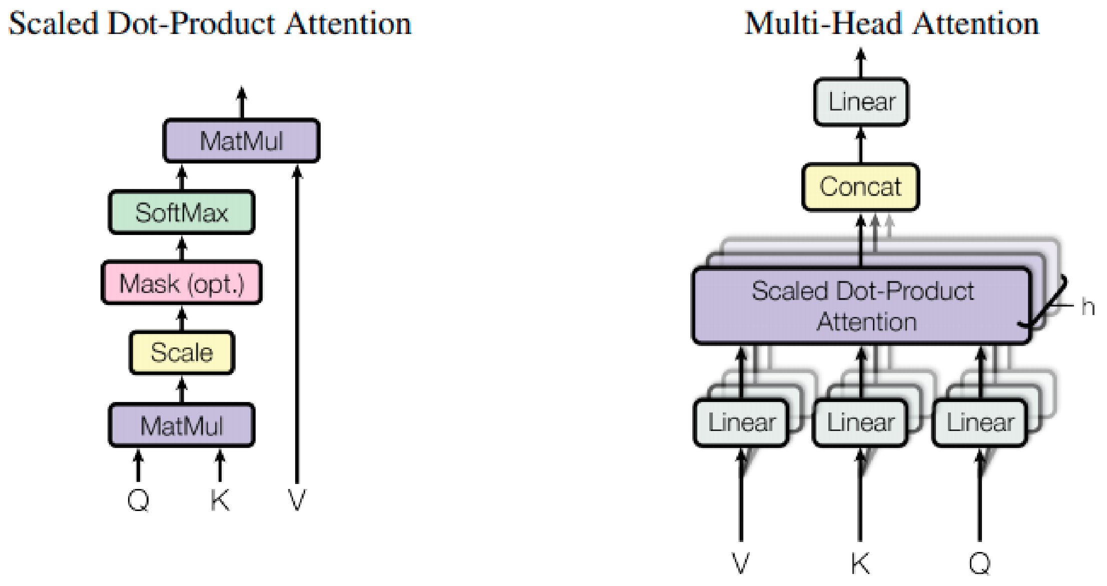


Figure 6: The Scaled Dot-Product attention and how it is used in the Multi-Head attention [7]

The operation of the multi-head attention is described in Equation 9 and Equation 10 where the matrices $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{model}}$ are parameter matrices that are updated during back-propagation. Essentially, a linear network, without bias. The matrices $\mathbf{V}$, $\mathbf{K}$, and $\mathbf{Q}$ have dimensions, ignoring batch dimension, $\mathbb{R}^{M \times d_{model}}$ where M is the length of the sequence, and d_{model} is the embedding dimension. The output from the scaled dot-product attention will have dimensions $\mathbb{R}^{M \times d_v}$. When concatenating the outputs from every head, h concatenations, the resulting dimensions are $\mathbb{R}^{M \times hd_v}$. After multiplying the concatenated matrix with the parameter matrix W^O, the result has dimensions $\mathbb{R}^{M \times d_{model}}$. Thus the result has the same dimensions as the inputs, with the values weighted based on the query and the keys.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \cdots, \text{head}_h)W^O \tag{9}$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{10}$$

The decoder in the Transformer architecture has a multi-head attention block that implements encoder-decoder attention. The attention mechanism simply has the output as the query and the keys, and the values are from the decoder itself. This results in the values in the decoder, which are past outputs from the whole transformer, are compared and weighted with the encoded inputs from the encoder. This is in contrast to self attention, which weights each part of the sequence's relation to another part of the same sequence

2.7.3 Positional encoding

Positional encoding is an essential part of the Transformer and allows the model to reason about the position of the inputs without the use of sequential mechanisms. It is very important to distinguish positional encoding from positional embedding, where the latter is a transformation with parameters that are learned and updated, while positional encoding is a normal transformation without parameters that get updated. There exist several different types of positional encoding, however, one of the simplest encodings used is the sinusoidal positional encoding. This is also the encoding used in the original Transformer paper [7]. The sinusoidal positional encoding follows the scheme seen in (11) and (12), where (11) is for even numbered indices in the input vector and (12) is for odd numbered indices.

$$PE(pos, 2i) = \sin(pos/1000^{2i/d_{model}}) \tag{11}$$

$$PE(pos, 2i+1) = \cos(pos/1000^{2i/d_{model}}) \tag{12}$$

The positional encoding gives each entry in the input vector a positional vector of the same length as the model size. The term pos indicates the position of the entry in the input vector, d_{model} is the dimension of the model, which determines the size of the positional encoding vector and therefore ultimately the number of unique positions that can be encoded. The i indicates the index in the positional vector. The positional vector encodes the position of the entry in input and thus the model can reason about the location of different elements relative to each other. Since positional encoding results in a vector for the position of every element in the input vector, the result is a positional encoding matrix. An illustration of this can be seen in Figure 7

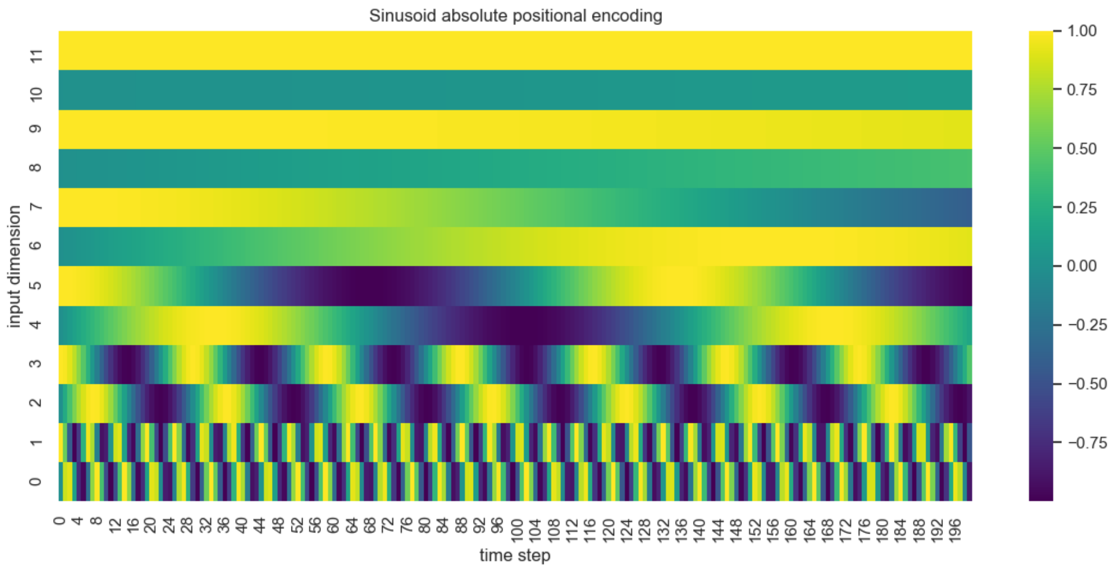


Figure 7: Illustration of sinusoidal positional encoding

This encoding is absolute, which means that an element in the encoded sequence will be assigned an index which is represented by a vector, this is in contrast to a relative positional encoding which represents the distance between the elements. How this is done is a bit more intricate. The relative encoding requires more memory since there exist $2N - 1$ differences between N elements. There are however advantages with the relative positional encoding. If the input to the model is of varying length, the model will have some bias towards the first elements in the sequence. The first elements would always be present in the training set, in contrast to the last elements in a longer sequence which are not always present. This might lead to the model focusing too much on the earlier samples. These encoding schemes are used both in the encoding of sequential data like text and speech, but also in spatial data like images.

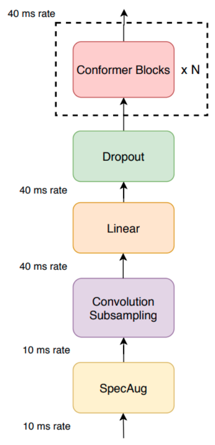
2.8 Conformer

The Transformer architecture has also been used for ASR. For instance [20] uses Transformers in their Transducer architecture, with both the acoustic encoder and prediction network being transformers. Both of these Transformers are equal in fact. Recent advancements have been made by extending and modifying the Transformer architecture for ASR. One of the more notable Transformer variants used for ASR is the Conformer. The Conformer presented in the original paper, [18], is used as an encoder network in a Transducer configuration with a single layer Gated Recurrent Units (GRU) prediction network [21]. The Conformer architecture addresses the limitations of

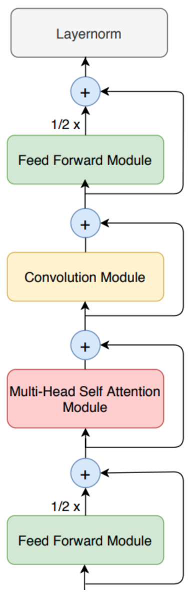
Transformers for ASR, which is their inability to extract local fine-grained details. This is however the specialty of CNNs. Thus, the Conformer introduces a special convolution block to capture these fine-grained local features.

Another distinction from the Transformer is how the skip-connections are used. The skip-connections are used to add residual information from a previous layer to the next layer by element-wise addition. In the Conformer model, these residuals are scaled by a factor of 0.5. Along with the scaled skip-connections, a sandwich-like structure with a feed-forward-network with the scaled skip connections before and after the multi-head attention mechanism is used. This sandwich-like structure was introduced with Macaron Net [22]. The idea behind this structure is to make the Transformer resemble the numerical solver for a Multi-Particle Dynamic System, which lead to the sandwich-like structure.

The convolution module used in the Conformer is placed after the multi-head attention in order to capture both global and local interactions between the speech features. The full conformer architecture is shown in Figure 8. On the left is the full architecture and on the right is the Conformer block which is introduced in [18]. The architecture that is depicted uses a spectrogram as input, which, during training is augmented with spectral augmentation [11]. The features are then reshaped and flattened, before being passed through a linear feed-forward network and the Conformer blocks. During training, dropout is used between the feed-forward network and the Conformer blocks to help with regularization.



(a) The conformer architecture



(b) The conformer block

Figure 8: Overview of the Conformer architecture on the left, and the components of a Conformer block on the right. Both taken from [18]

The feed-forward module in the conformer follows the same structure as the feed-forward module in [20]. This is shown in Figure 9.

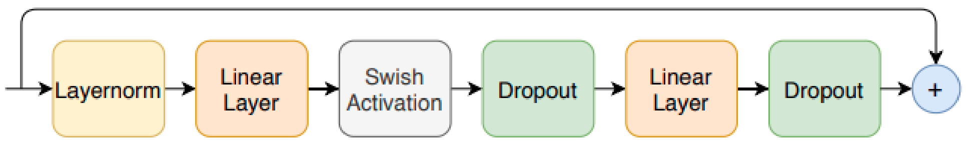


Figure 9: The feed-forward module of the conformer network. Taken from [18]

The only difference between the two is the activation function after the first linear layer, which is changed from Rectified Linear Unit (ReLU) to Swish.

$$f(x) = \frac{x}{(1 + e^{-x})} \tag{13}$$

The equation for the Swish activation function is shown in (13). The difference between the ReLU and the Swish activation function is plotted in Figure 10, from this it is clear that the Swish is not monotonically increasing and is also a smooth function in contrast to the ReLU.

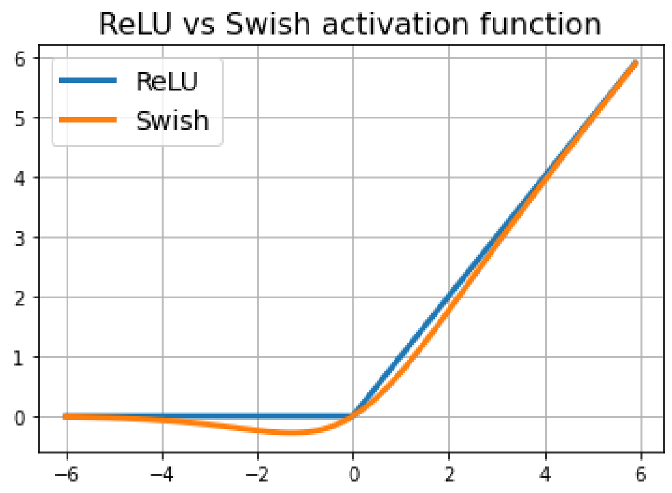


Figure 10: The ReLU and Swish activation functions.

This smoothness helps the model at converging towards a minimum. Also, the Swish activation function is nonzero for small negative values whereas the ReLU is cut of at zero. Being bounded below, like both the ReLU and the Swish functions are, helps regularization, but some small negative values may be beneficial in training [23].

The Layernorm block is a layer normalization [24], and is used on the input tensor going into each of the modules in the Conformer blocks. This is used to help the model generalize better and reduce training time. The Layernorm normalizes the input data, $\mathbf{x}$, and then applies a scaling and shifting step with learned parameters γ and β. Given an input vector $\mathbf{x}$, the normalization is done by first finding the mean and standard deviation by the use of formulas 14 and 15. The normalization is done via equation 16. The ϵ is a small scalar used to avoid accidental division by zero. Finally, the scaling and shifting are shown in equation 17. The superscript l denotes the layer, and K denotes the number of elements in the vector. If the input has more than one dimension, the summation is done over the entire tensor. In the original paper, the layer normalization is done a bit differently than the implementation in the Conformer where it is taken over over the summed outputs from each hidden layer in a network.

$$\mu = \frac{1}{K}\sum_{i=1}^{K} x_i \tag{14}$$

$$\sigma = \sqrt{\frac{1}{K}\sum_{i=1}^{H}(x_i - \mu)} \tag{15}$$

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{16}$$

$$y = \hat{\mathbf{x}}\gamma + \beta \tag{17}$$

The convolution module of the conformer is shown in Figure 11. This module is responsible for capturing local features in the data.

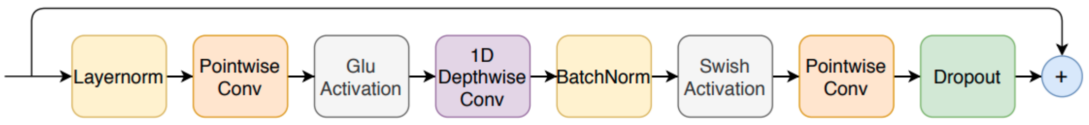


Figure 11: Convolution module of the Conformer. Taken from [18]

The inputs are first passed through a layer normalization. Next, a Pointwise Convolution is used. The Pointwise convolution is a convolution with a 1×1 kernel with stride 1 and has a depth equal to the number of channels in the input. So, for instance, an image with three channels that undergoes a Pointwise convolution to produce a six-channel output would have six 1×1 kernels with stride 1. After the Pointwise convolution, the resulting feature maps then pass through a Gated Linear Unit (GLU), which acts as a control gate, restricting the amount of information that passes on to the next layer. The GLU function is shown in (18), where a and b are the first and the second half of the input. The split is done over the last dimension of the input. The operator $\otimes$ is the Hadamard product and is essentially element-wise multiplication between the elements in the tensors. σ is an activation function.

$$\text{GLU}(a, b) = a \otimes \sigma(b) \tag{18}$$

Next is the 1D Depthwise Convolution, which is a little bit different from the Pointwise convolution. The input channels each have their own kernel that is used only on that specific channel. The implication of this is that the number of output channels must match the number of input channels. Next, a BatchNormalization layer is used, which works in almost the same way as the layer normalization. However, the layer normalization is used on each batch independently, while batch normalization normalizes the elements of an input over the batch dimension.

The final modules of the convolution module are a Swish activation, followed by another Pointwise convolution and a dropout layer at the end for better regularization.

The multi-head self-attention module has a quite simple layout, a layer normalization is applied before the multi-head self-attention and a dropout is again applied at the output. This is illustrated in Figure 12. Since the Conformer is supposed to work with different lengths of utterances and has an input that is relatively long, compared to normal text. The TransformerXL [25] style multi-head attention was incorporated in the design of the Conformer. This includes the relative sinusoidal positional encoding scheme. This positional encoding is introduced in the hidden state in the self-attention as opposed to the original Transformer, where the positional encoding is introduced at the input. The derivation of this positional encoding scheme is rather extensive [25].

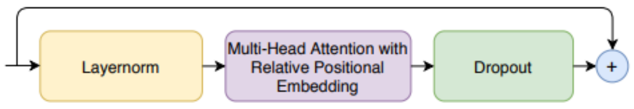


Figure 12: the multi-head self attention block in the Conformer. Taken from [18]

As mentioned earlier, the original paper used the Conformer in a Transducer configuration with a single layer GRU-cell predictor network. The GRU-cells are a type of recurrent units, meaning that they use past outputs from the unit for prediction at the current time step. This is comparable to giving the neural network memory. The GRU-cell is most comparable to the Long Short-term

Memory (LSTM) architecture. Both of them are recurrent, but the main difference is that the GRU-cell lacks an output gate and thus has fewer parameters. Thus, the GRU only has two gates, an update gate and a reset gate. They are quite intuitive in that one decides what information to remember and the other decides what information it should forget.

3 Confidence Estimation

When predicting or classifying anything with the use of neural networks, it is beneficial and in many situations, crucial to get an estimate for the confidence of the model. How sure is the model that the prediction is correct? This might seem like a trivial task, by using the probabilities given by the softmax activation. However, it is not that straightforward. Consider the case of a three-class classifier. Since the softmax produces probabilities that would sum up to one, there would a minimum of 1/3 probabilities for at least one of the classes. And that is even if the model gets fed a sample that does not belong to any of the classes. In other words, the model has no way of saying "I do not know", or how sure it is. This feature can be quite powerful when it comes to certain applications. In some applications, this feature is necessary, for instance in voice assistants. The ability for a voice assistant to respond back to the speaker and ask for a repetition of the voice command enhances the usability of the system, rather than the voice assistant using a low confidence command. Secondly, some transcription tasks require a high degree of accurate transcriptions, such as spoken medical journals or transcription from trials. Providing a confidence measure for transcriptions will allow for low-confidence words to be highlighted and can easily be corrected by humans. The confidence measure can also be incorporated into the training loop and allow for better-performing models. In the case of self-supervised learning, high-confidence transcriptions can be used to train the model. This can also be used in a student-teacher setting with an unlabeled dataset.

A confidence measure can also be incorporated in a Beam Search algorithm during decoding or for rescoring methods. This will in theory decrease the Word Error Rate (WER) of an ASR system. By dynamically adjusting the weight of the LM based on the confidence for a word, more use of the LM can be used for words that are hard to hear or understand for the acoustic encoder. Conversely, if the acoustic encoder is confident in its prediction, a strong influence from the LM would be disturbing. In spontaneous speech, people may tend to speak in incomplete sentences or use a sequence of words that is unlikely if it was a written sentence. This dynamic weighting would in theory decrease the Word Error Rate (WER) of the ASR system in these cases. In addition, dynamically relying more on the acoustic predictor and possibly eliminating the LM if the confidence for a word is high, the runtime can be improved.

3.1 Monte Carlo Dropout

In order to provide a confidence estimation there exist a few tried and tested solutions. Some are network architecture agnostic, while some are tailored to specific networks or tasks. One of the more basic and architecture agnostic approaches is Monte Carlo dropout. This approach, as the name suggests, relies on dropout. This method is relatively simple to implement and understand, additionally, it can be applied to any existing network. During the training of neural networks, dropout between layers is a common technique to improve generalization and prevent overfitting. During inference, the dropout is normally not used in order to use the full network and thus get a better prediction. If the dropouts are however also activated during inference and the result from multiple inferences are sampled, a distribution of the different predictions can be used to device a confidence estimation. This is done by measuring the predictive entropy of the output predictions. Entropy is a measure of the order or disorder in a system. The predictive entropy is defined as shown in 19, from [26]

$$\mathbb{H}[y|\mathbf{x}, D] := \sum_{c} p(y = c|\mathbf{x}, D) \log p(y = c|\mathbf{x}, D) \tag{19}$$

This equation can make little sense at first glance, however, if this definition is first derived from the definition of surprisal, shown in Equation 20, the information it captures is easier to interpret.

$$I(x) = \frac{1}{\log(p(x))} \tag{20}$$

This metric gives an indication of how surprising it would be for an event of probability p to occur. In a sense, this is the inverse of the probability, but not quite, since there is a logarithm involved in the definition. The predictive entropy, sometimes referred to as Shannon entropy, is the expectation value of the surprise. This is shown in 21.

$$\begin{aligned} \mathrm{H}(X) &= \sum_x -p(x)\log p(x) \\ &= \sum_x p(x)\mathrm{I}(x) \\ &\overset{\text{def}}{=} \mathrm{E}\left[\mathrm{I}(X)\right] \end{aligned} \tag{21}$$

This means that the predicted entropy is the expected surprisal. Because of this, the predictive entropy captures the average information that the predicted distribution contains. The average information will attain its maximum value if all classes in the predicted distribution have the same probability. Conversely, the minimum value is attained if one class has a probability of one and all other classes have zero probability.

In Equation 19, $\mathbf{x}$ is the input vector to the model from the dataset D, y is the prediction and c is a class that the prediction can take.

In the case of Monte Carlo dropout, the predictive entropy is estimated by taking T forward passes with the same input vector $\mathbf{x}$. Then, for each class c, averaging the probabilities from that class in the T probability vectors. This is then the estimate for the probability used in 19. In other words, $p(y=c|\mathbf{x},D)$ is replaced with $\frac{1}{T}\sum_t p(y=c|\mathbf{x},D)$. This results in the following equation:

$$\mathbb{H}[y|\mathbf{x},D] := \sum_c \Big(\frac{1}{T}\sum_t p(y=c|\mathbf{x},D)\log\frac{1}{T}\sum_t p(y=c|\mathbf{x},D)\Big) \tag{22}$$

The simplicity of the Monte Carlo dropout technique is the biggest selling point of the technique.

3.2 Deep Ensembles

Another somewhat similar approach to the Monte Carlo dropout is the use of Deep Ensembles [27]. This approach makes use of multiple different neural networks to produce a prediction. This approach is not just for estimating the uncertainty of a prediction, but also to boost performance. There exist several different ensemble methods. For instance, a Mixture of Experts model is an ensemble model, which consists of several different models or modules that are experts at something. This could be a convolution module or a MLP module. They are combined with gating or routing mechanisms, such as the Switch Transformer [3]. Deep Ensembles have no such gating or routing mechanisms. Instead, the parallel models are joined to produce a single prediction. The confidence can be estimated with an additional network that can estimate the uncertainty based on the input from the ensemble of models, like in [27]. They use a network with two outputs, one for mean and the other for variance, as the final layer of the model. The variance will then be used as a measure for the confidence the model has in its prediction. In the case of [27] the negative log likelihood criterion in 23 is used to train this final layer.

$$-\log p_\theta\left(y_n \mid \mathbf{x}_n\right) = \frac{\log\sigma_\theta^2(\mathbf{x})}{2} + \frac{\left(y-\mu_\theta(\mathbf{x})\right)^2}{2\sigma_\theta^2(\mathbf{x})} + \text{ constant} \tag{23}$$

In this equation, $\mathbf{x}$ is the output vector from the ensemble of vectors, y is the label and (.) and $\sigma(.)$ are functions for the empirical mean and variance. One could theoretically also use the predictive entropy as a metric for the uncertainty, as in the Monte Carlo Dropout method.

The models used in Deep Ensemble models is not restricted to be the same model, but can be models that are different in architecture but perform quite equally. They can also be the same model, that has been trained under different conditions. Either different dataset or using different hyper-parameters.

This type of method can not only produce a quantity for the uncertainty of the predictions, but it can also help reduce the uncertainty. When developing a neural net for a set of models may be produced with different architecture or variations in some layers. Some of them perform better at certain parts of the validation set, or they perform unevenly. Instead of throwing away all of the models and selecting the marginally better performing model, they can be put together in a Deep Ensemble model. The same model initialized with a different seed will converge with a different set of parameters, while having virtually equal performance on a training set [28]. During inference, however due to this difference in parameters the models might perform differently on a test set since the test set slightly differs from the training set. This difference in prediction will allow for the calculation of a confidence estimate.

3.3 Bayesian Neural networks

In classic neural networks, the weights in a layer have a scalar value which is used to generate a single latent representation that gets passed on to the next layer. In a Bayesian neural network, the weights can attain many different values since they are associated with a distribution. The mean of this distribution would be the weight that the network is most confident in being the correct one. In order to get a confidence estimation from this type of network, one would need to sample from the weights and biases a multiple times and use these to produce predictions. This will yield several predictions with slightly different output distributions, just like the previous methods.

Obtaining weights as part of a distribution is a somewhat tricky process. In order to find these weights, also known as posterior densities, the equation is shown in Equation 24 is used. Here the inputs are denoted as x and the outputs are denoted as y. The weights are denoted as w.

$$p(\mathbf{w} \mid \mathbf{x}, \mathbf{y}) = \frac{p(\mathbf{x}, \mathbf{y} \mid \mathbf{w}) p(\mathbf{w})}{\int p(\mathbf{x}, \mathbf{y} \mid \mathbf{w}) p(\mathbf{w}) d\mathbf{w}} \tag{24}$$

The posterior $p(\mathbf{w})$ relates to how the weights are distributed before the model has seen any data. The $p(\mathbf{x}, \mathbf{y} \mid \mathbf{w})$ describes how the data might be distributed.

Since the denominator contains an integral that sums over all the possible values that w can attain, this method is quite hard to do. There are ways to circumnavigate this issue however. For instance, a Markov Chain Monte Carlo approximation can be used, this involves a set of algorithms that constructs a Markov chain in order to do Monte Carlo approximation. This is a not very practical method since it is hard to make the model converge and the method is computationally expensive [29]. Therefore, details of the algorithm will not be discussed further. See [29] for further details.

Variational Inference is another approach of estimating the denominator of Equation 24 which is better for deeper neural networks. In this approach, a distribution is chosen and changing the parameters that define this distribution such that it closely matches the density we want to estimate.

3.4 Confidence Estimation Module

The last approach is directly targeted toward speech recognition and is not directly applicable toward other tasks or architectures. This method involves using an additional neural network that is independent from the ASR network. There has been a few attempts at developing this module, often referred to as a Confidence Estimation Module (CEM) [4], [30]. A conceptual illustration of this is shown in Figure 13

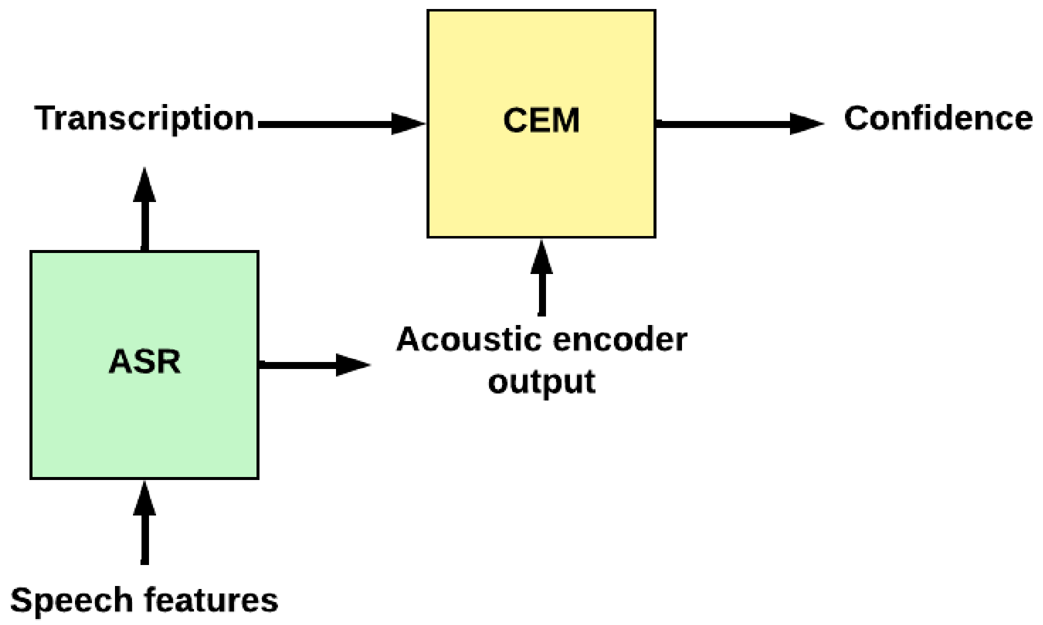


Figure 13: Overview of how a Confidence Estimation Module would work together with an ASR system.

This module generally uses the latent representation from the acoustic encoder and the token predictions at the current time step as inputs. There are however some differences in what features are used, this also depends on the architecture used. CEMs has been implemented with different architectures, both when it comes to the architecture of the CEM and the ASR module. The module is trained with a frozen ASR network, with the features being generated on the fly. The CEM uses features generated from the transcription network from the time step in which a token is emitted. The labels used in the CEM are usually binary, [4], [30] on the token or word level.

The main advantage of the CEM compared to the other methods is that the model only requires a single forward pass in order to produce a confidence measure for a token. The other methods generally require several forward passes from the transcription model in order to produce a confidence estimate. Of course, the methods can easily be parallelized, but at a quite significant memory requirement since the transcription networks can be quite large. The CEM can also be significantly smaller than the transcription network, [30], thus reducing the memory requirement for the confidence estimation as opposed to the other methods presented.

4 Methodology

4.1 Conformer

The ASR model used in this thesis, which will be used as a basis for the confidence estimation, is a Conformer model [18] in a Transducer configuration. This model was chosen since it is quite easy to align the predictions with the corresponding speech features in the decoding algorithm. The model has the following relevant configurations as shown in Table 1. The overall model architecture is shown in Figure 14. This model was developed in connection with [6].

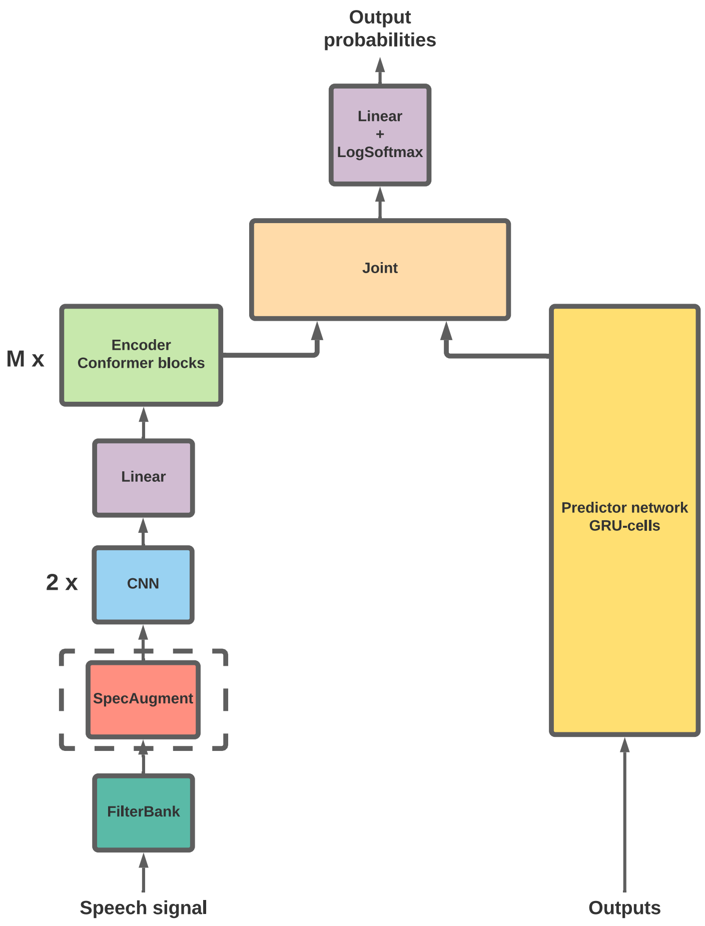


Figure 14: The Conformer Transducer model. The SpecAugment model is only used during training.

Table 1: The relevant specifications of the Conformer Encoder.

Parameter	**Value**
Model dimension	256
Attention heads	4
Layers	16
Kernel size	31
# Parameters	25.6M

The Conformer Transducer has a single layer GRU predictor network with 640 cells. This addition and the joint network, puts the total parameter count for the Conformer Transducer at 42.2 million.

The model is trained with the Adam optimizer and follows the same learning rate schedule as in the original Transformer paper [7]. The equation describing the learning rate as a function of the step number is:

$$\mathrm{lr} = \min(\mathrm{step\_num}^{-0.5}, \mathrm{step\_num} \cdot \mathrm{warmup\_steps}^{-1.5}) \cdot d_{\mathrm{model}}^{-0.5} \tag{25}$$

d_{model} refers to the model size, *warmup_steps* is the number of warm-up steps before the maximum learning rate is achieved. This figure is set to $8,000$.

The performance of these models is quantified using two metrics, the Word Error Rate (WER) and the Character Error Rate (CER).

$$WER = \frac{S_w + D_w + I_w}{N_w} \tag{26}$$

$$CER = \frac{S_c + D_c + I_c}{N_c} \tag{27}$$

As the names suggests, these metrics capture the rate of error in a sample. The definitions of these metrics are shown in Equation 26 and Equation 27. The metrics are very similar and differ only on which level they operate. The WER operates on the word level and the CER operates on the character level. In the equations, the N is the length of the sequence, S is the number of substitutions necessary to produce a correct sequence. D and I is the necessary deletions and insertions necessary to produce the correct sequence. The metrics treat every mistake equally, which means that an insertion error is as costly as a deletion error. While in reality, a deletion on word level might be more semantically wrong than an insertion. This might be true for the inverse in some cases as well. These metrics are convenient for comparing similar models. However, the true performance and the types of mistakes the model makes are harder to capture in a metric or a set of metrics. Thus an inspection of the test samples is necessary to determine the types of errors the model produces and the usability of the model. The discrepancy between the CER and WER can give an insight to these types of errors. If the CER is significantly lower than the WER, this gives an indication that the model make small errors like spelling or grammatical errors. This is however no substitute for the time-consuming manual inspections.

The model that is used as a basis for the confidence estimation has a Word Error Rate (WER) of 29.41%. This model was chosen instead of a Sequence-to-Sequence (Seq2Seq) Transformer model which was also developed in the specialization project [6]. The Seq2Seq model was larger and also achieved a lot better performance on the test set with a WER of 14.78%. There is several reasons for this decision. Firstly, the model is quite a lot larger, with 80.8 million parameters, compared to the 42.2 million parameters of the Conformer Transducer. This reduces the computation time for a single forward pass and the memory requirements. The second reason, and main reason for this choice is that during decoding, the emitted tokens can be associated with a specific time step. For the Seq2Seq Transformer, this is not a very straightforward process as the decoder can use the whole acoustic context for decoding a single token. Using the whole speech sample to estimate the confidence of a single token is not very efficient. Training might be very tricky since the model then has to be able to align features with tokens, and learn to estimate the confidence. Thus, being able to extract just the acoustic features that make up a token and only use these during the confidence estimation, is very efficient and beneficial.

4.2 Norwegian Parliament Speech Corpus

As mentioned previously, the data used for training the Conformer Transducer is the Norwegian Parliamentary Speech Corpus (NPSC) from the Norwegian parliamentary.

The dataset used consists of 140 hours of transcribed Norwegian speech. This data consists of several speakers with varying dialects. The data is segmented into shorter wav files with lengths from a few seconds, up to just under a minute. In Figure 15 a histogram is plotted with time intervals on the x-axis and the number of wav files with the corresponding duration is on the y-axis. The majority of the speech samples is under 15 seconds long, however, there is still a significant portion of the total speech that is over 15 seconds long, since the files are longer in duration even though there are few of them.

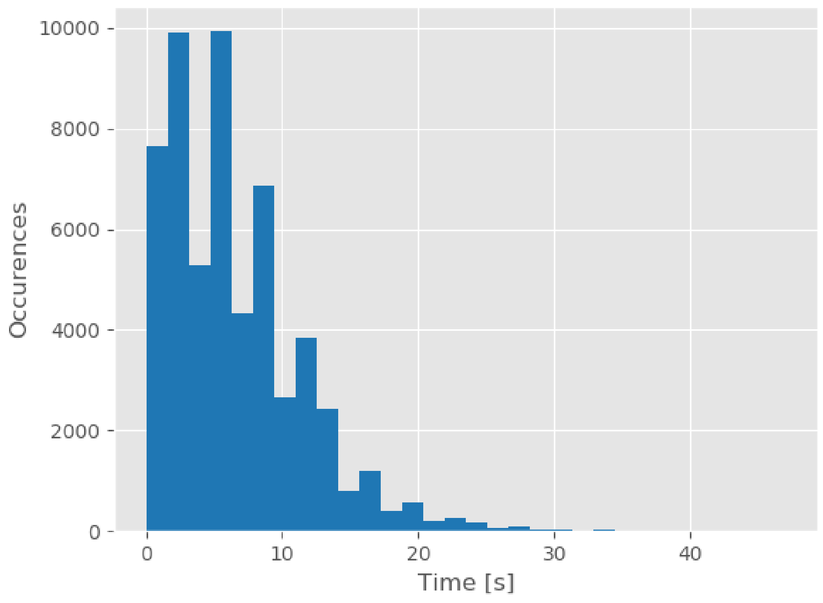


Figure 15: Histogram showcasing the distribution of the duration of data in the NPSC dataset

Of the total of 140 hours in the corpus the data is split into three segments. One segment for training, one for validation, and one for testing. These segments consist of 91.8 hours, 9.1 hours and 9.0 hours respectively. Some of the data is lost due to the cut-off at 15 seconds, and some are lost when splitting the full recording of the meeting into samples. The dataset includes a mix of two written languages, Bokmål and Nynorsk. The Bokmål transcriptions make up the majority of the corpus with 87.2%, resulting in the Nynorsk part only accounting for 12.8%. This, along with some other statistics is showcased in Table 2.

Table 2: NPSC corpus statistics from [31]

Duration, pauses included	140.3 hours
Duration, pauses excluded	125.7 hours
Word count	1.2 million
Sentence count	64 531
Language distribution	Nynorsk: 12.8% Bokmål: 87.2%
Gender distribution	F: 38.3%, M: 61.7%

A notable remark from the statistics is the amount of pause in the dataset, which is 14.6 hours, or 10.2%. This is due to a lot of pauses, vocal and nasal hesitations, which is most likely a result of the speakers not reading from a manuscript. This finding is also consistent with the observations done by listening through parts of the dataset.

Some of the speakers contribute significantly more than others to the training set. This is evident in Figure 16, which shows the number of samples in the training set for each speaker. This might introduce some biases favoring these speakers and their characteristics.

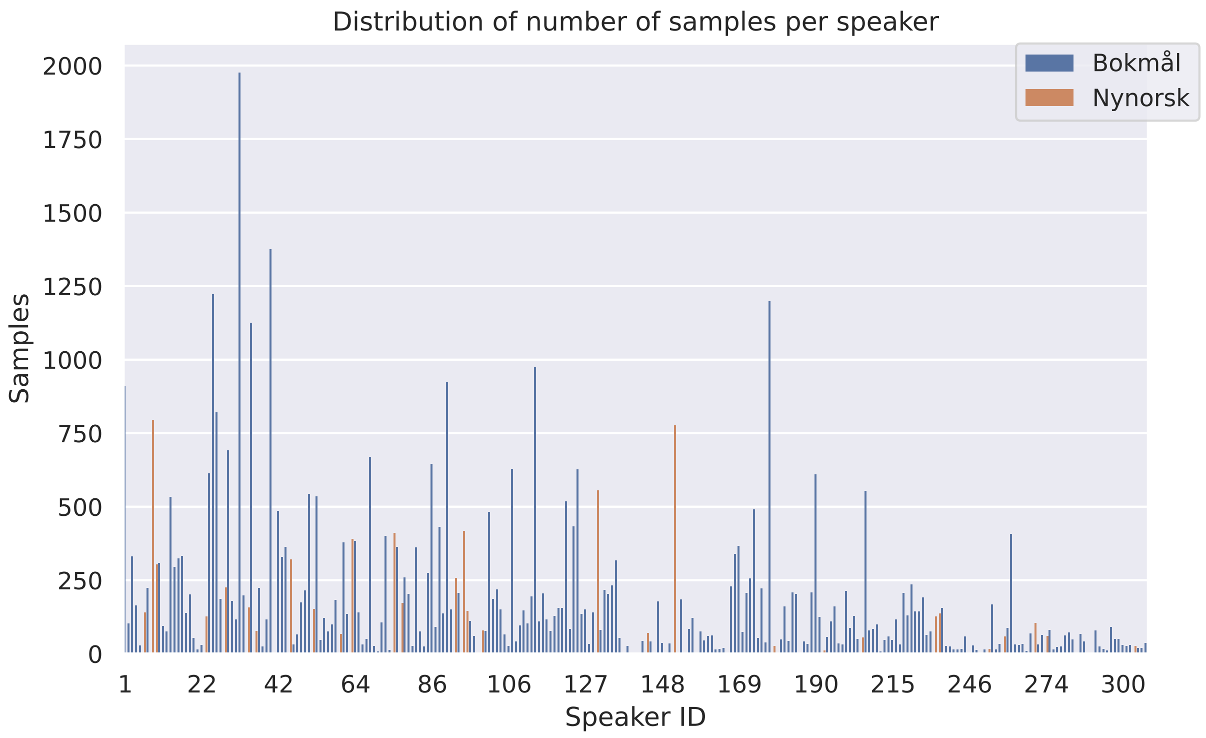


Figure 16: Distribution of the number of samples per speaker in the NPSC dataset. The color of the bar denotes the written language. Blue is Bokmål, orange is Nynorsk.

For the ten speakers with the most samples, only one speaker is transcribed with Nynorsk, and this speaker has the least amount of samples of the ten. This distribution is shown in Figure 17. This figure shows the number of samples with a given duration as the width of the bar corresponding to the speaker id. From the figure, it is clear that some of the speakers have a lot of very short samples. These samples are also likely very similar in content. Since the recordings are from very formal meetings in the parliament, a lot of the samples are people addressing other people or making announcements.

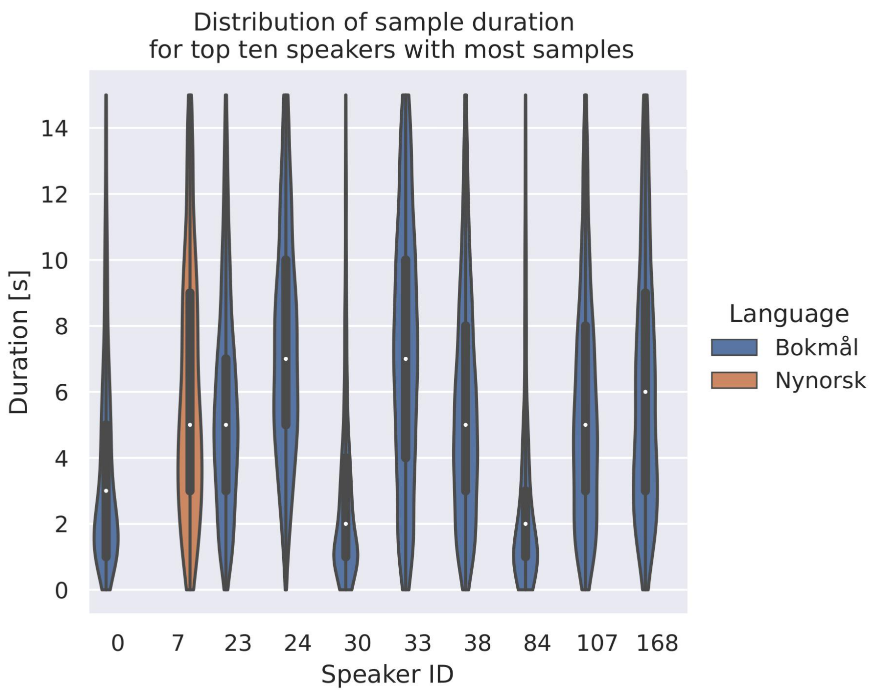


Figure 17: Distribution of sample duration for the ten speakers with the highest total amount of speech. The color of the bar denotes the written language. Blue is Bokmål, orange is Nynorsk. The width of the bar represents the number of samples with the corresponding duration shown on the y-axis.

The dataset also contains information about where the speakers are from. The geographic origin of the speakers are described on two levels, region and county. This likely has an influence on the speaker's dialects. The number of samples per region in the training dataset is shown in Figure 19. The number of samples per county in the training set is shown in Figure 18. Some of the speakers are missing the region and county identifier and is thus listed with "Unknown".

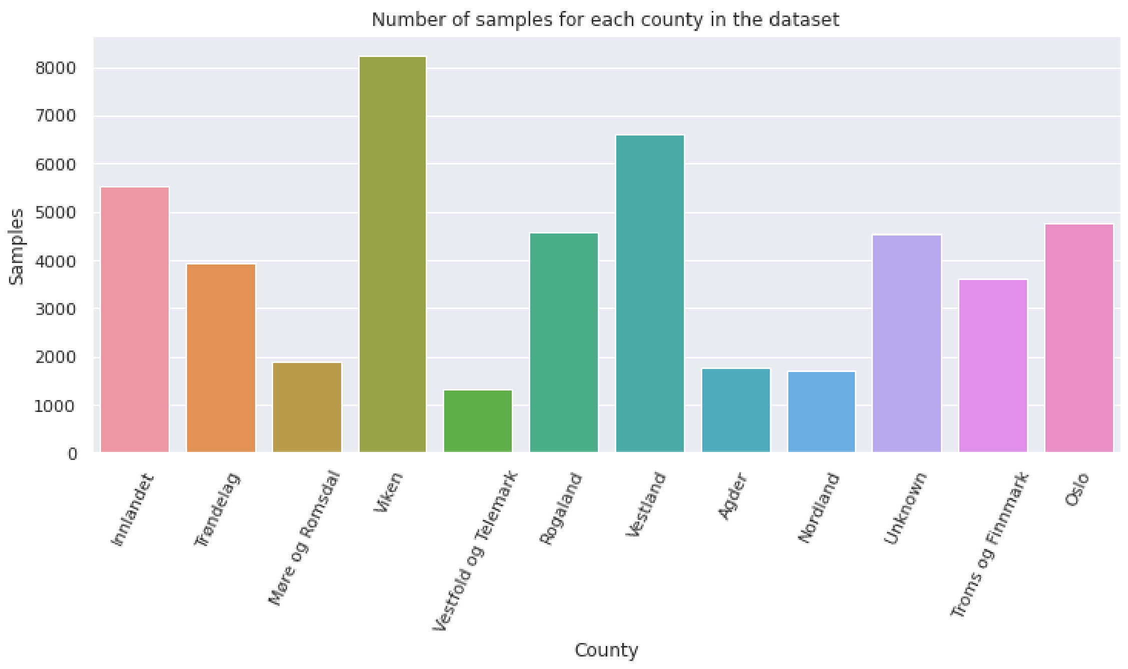


Figure 18: Bar-plot showing the number of samples on the y-axis and the corresponding county on the x-axis.

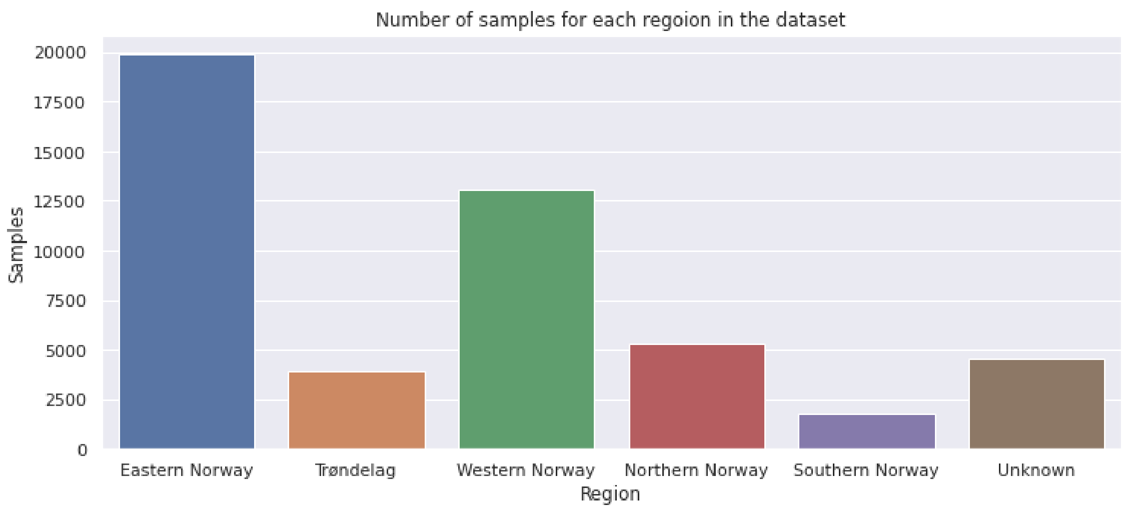


Figure 19: Bar-plot showing the number of samples on the y-axis and the corresponding region on the x-axis.

The regions “Eastern Norway” and “Western Norway” make up a significant portion of the training set compared to other regions, thus one might argue that this introduces certain biases towards these regions.

4.3 Confidence Estimation

For the confidence estimation, a Transformer model is developed. This model should be advanced enough to be able to estimate the confidence of the ASR system quite well. This is the same type of architecture used in [30]. The self-attention will also help in determining the lack of similarity between the predicted token, and what the acoustic encoder “hears”.

The features for this model are generated by using a Greedy Search algorithm with the previously mentioned Conformer Transducer. When a token is emitted, the latent representation from the acoustic encoder and the token id is fed to the model. This is the features that the CEM uses to produce a confidence estimation. The token id is first passed through an embedding which is then concatenated with the acoustic features.

The features from the prediction network are not used for the confidence estimation. This is done because the acoustic encoder is the main component in the ASR system and its confidence is what should be in focus. The prediction network is an LM which is making educated guesses for the next token. Thus, the confidence of the prediction network is not really interesting.

Since the models are supposed to produce a confidence estimation for the current predicted token, the output from this model is a single neuron with a sigmoid activation function. This activation function is different from a ReLU or a Softmax. The ReLU activation function, and its relatives (GELU, SiLU, etc.) do not have an upper bound and some can also produce negative numbers, such as GELU. The Softmax, which is normally used for classification tasks is:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{28}$$

This activation function requires multiple output nodes as the function creates a probability for each of the nodes. This means that the sum of the scalar outputs at the nodes sums up to one. If the output is a single node, then the value of this node will always be one.

The sigmoid function is bounded above at one and has a lower bound at zero, thus it is ideal for providing a single scalar value that measures the confidence of the transcription network. This means that the labels for these models will have to be within the same range as the sigmoid function, which is zero to one.

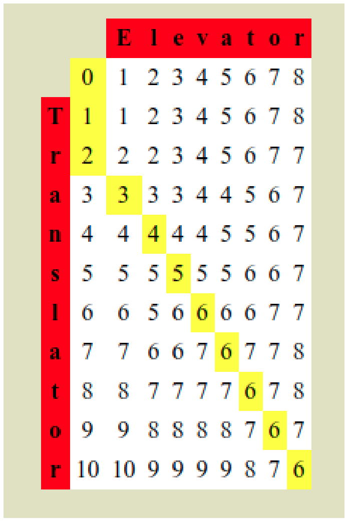
These labels are generated by first doing a full Greedy Search of a speech sample and thus obtaining the transcription. Then this transcription is compared with the ground truth transcription. The predicted words that do not match the ground truth are labeled as wrong, 0, and the ones that match are labeled as correct, 1. Then the latent representation from the acoustic encoder associated with this word along with the id's is fed to the CEM. The confidence is then calculated and the model is updated based on the loss. This is pretty straightforward in theory, however, there are some elements to this that complicates things. Firstly, the ground truth labels are not time aligned, which means that the time step of when a word is spoken is not known. This means that the full prediction of a speech sample is needed before doing a comparison between the prediction and the ground truth. Secondly, the predictions might be completely wrong, having fewer or more predicted words than what is actually spoken. This means that it is not possible to simply compare each of the words sequentially. Instead, an alignment algorithm has to be used to find the alignment between the ground truth and the hypothesis and figure out if a word is correct, wrong, missing, or out of place.

4.3.1 Alignment

The alignment algorithm used is called Wagner Fischer and uses dynamic programming to compute the edit distance between the words in the ground truth and the hypothesis. The edit distance used is the Levensthein distance. This distance is expressed as a function in Equation 29, where *tail*(.) is all but the first word of the string that is input to the function.

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0 \\ |b| & \text{if } |a| = 0 \\ \operatorname{lev}\ (\operatorname{tail}(a), \operatorname{tail}(b)) & \text{if } a[0] = b[0] \\ 1 + \min \begin{cases} \operatorname{lev}(\operatorname{tail}(a), b) \\ \operatorname{lev}(a, \operatorname{tail}(b)) \\ \operatorname{lev}(\operatorname{tail}(a), \operatorname{tail}(b)) \end{cases} & \text{otherwise} \end{cases} \tag{29}$$

The Levensthein distance is the lowest cost of operations needed in order to convert a sequence A to another sequence B. The result from the Wagner Fischer algorithm is a matrix with the corresponding edit operations needed to convert sequence A to sequence B. An example of this matrix is shown in Figure 20. Here, the Levensthein distance is found between two words and finding the operations needed to convert ”Elevator” to ”Translator”. In order to find the lowest cost edit operations, a backtracking algorithm needs to be applied to the matrix. By starting at the lower right corner and moving to the upper left corner and choosing the lowest number, this cost is found. When moving, if the current letter in the words does not match each other then it needs to be replaced, inserted, or deleted. This operation is determined by the cost of the operation. If the lowest cost operation is moving to the left then it is a deletion, up is an insertion, and diagonally is a replacement, if the letters corresponding to that cell do not match.



		E	l	e	v	a	t	o	r
	0	1	2	3	4	5	6	7	8
T	1	1	2	3	4	5	6	7	8
r	2	2	2	3	4	5	6	7	7
a	3	3	3	3	4	4	5	6	7
n	4	4	4	4	4	5	5	6	7
s	5	5	5	5	5	5	6	6	7
l	6	6	5	6	6	6	6	7	7
a	7	7	6	6	7	6	7	7	8
t	8	8	7	7	7	7	6	7	8
o	9	9	8	8	8	8	7	6	7
r	10	10	9	9	9	9	8	7	6

Figure 20: The Levensthein edit distance matrix for the words Elevator and Translator

When this backtracking is done, a sequence of operations of the necessary operations needed to convert “Elevator” to “Translator” is obtained. And it is this sequence that is used to make the labels for the CEM. By using the aforementioned method on a word level for the ground truth and the hypothesis, the alignment problem is solved. The correct words are labeled as a one, and the words that should be edited, substituted, or deleted are noted as a zero. Insertions, that is a word in the hypothesis is missing, is not possible to label since it is impossible to find the emission time for a word that is not emitted, they do not exist.

The hypothesis is produced autoregressively on a token level, meaning that even though the labels are associated with a full word, the model outputs sub-words. In order to match the labels with the sub-words, one could try splitting the ground truth and hypothesis into sub-words and doing the alignment based on a sequence of sub-words. This will not work however, since the tokenization is non-monotonic, meaning that a word can be constructed several different ways with different sub-words. Instead the token output from the ASR needs to be associated with the words and then the corresponding labels in order to label the individual tokens.

4.3.2 Model

The Transformer model used as the CEM, is in itself quite simple in that it consists of N encoder layers from the original Transformer [7]. The output from the Transformer is then fed to a two layer MLP with a ReLU activation function in between before being output using the sigmoid activation function. This MLP is used to reduce the feature dimension from the dimension used in the Transformer to a single scalar that is used as the confidence score. The model structure is depicted in Figure 21

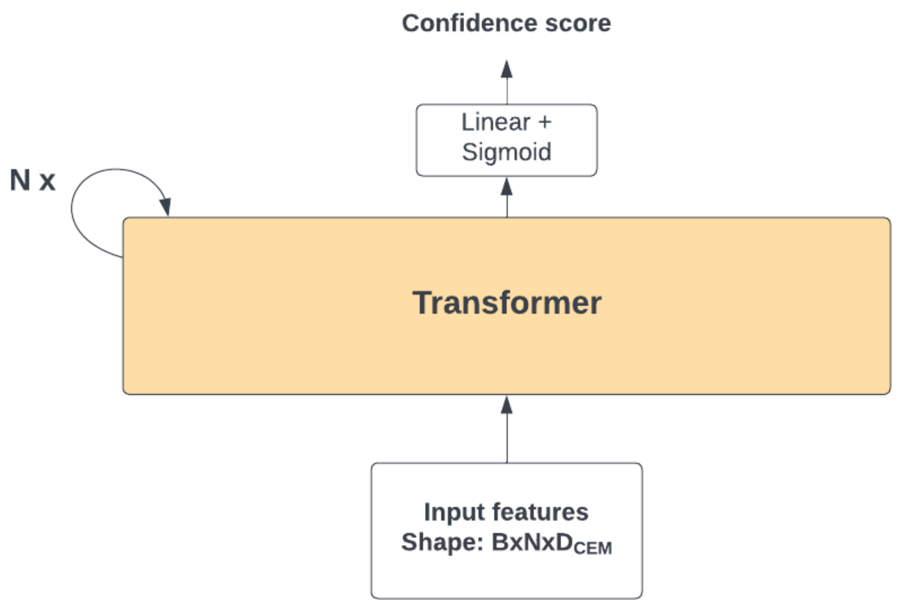


Figure 21: CEM architecture

The Transformer uses features from the acoustic encoder and the id of the predicted token. This id is fed into an embedding layer with the same dimension as the model dimension in the Transformer. By using an embedding, it is easier for the model to distinguish between different tokens, than if it was just a single scalar value. The acoustic features used are from the latent representation of the acoustic encoder. When a token is emitted, the current latent representation, as well as the latent representation from the two previous time steps are also used. These three tensors are then concatenated. By flattening and concatenating along the model dimension, the resulting tensor has effectively a length of one instead of three and a model dimension three times that of the acoustic encoder. These acoustic features and the token id-embedding are then concatenated to make a sequence of length two. This is all depicted in Figure 22. An absolute positional encoding scheme is then applied to these features before being input to the CEM.

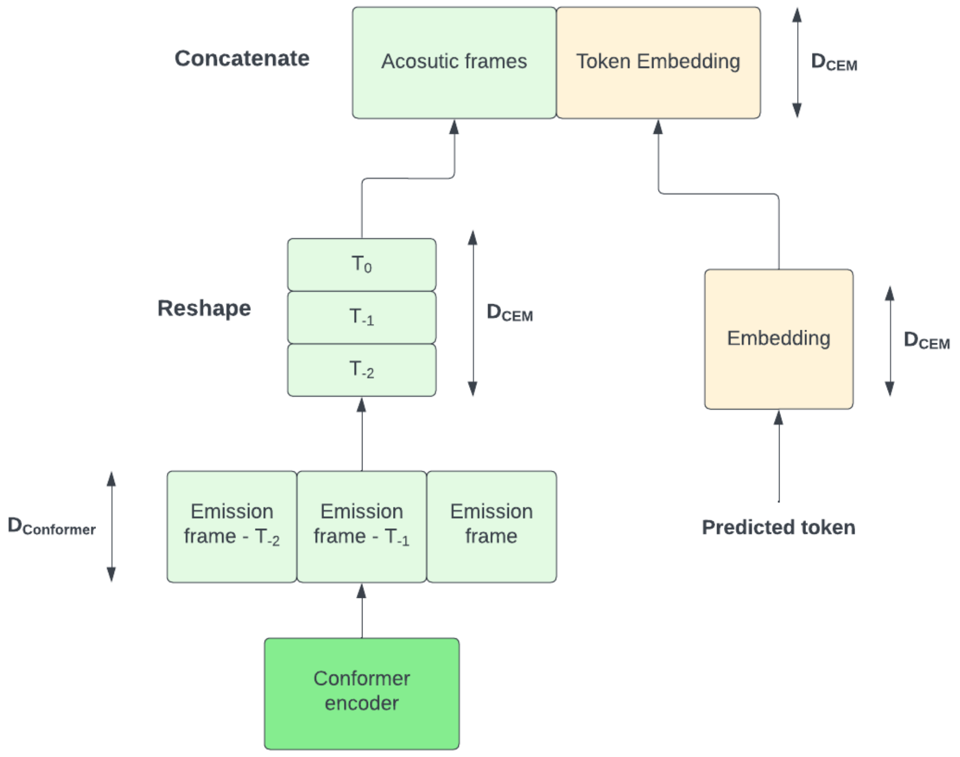


Figure 22: Illustration of features used in the Confidence Estimation module

The CEM has $N = 4$ layers and since the model uses a total of 3 acoustics frames that are flattened, this results in a model dimension of:

$$d_{CEM} = 3 \cdot d_{model} = 3 \cdot 256 = 768 \tag{30}$$

The number of acoustic frames has a direct influence on the model size, therefore only three frames was chosen. Also, no significant performance gain was found in [30] when using more frames.

This is will also have to be the dimension of the token embedding. This results in the total number of parameters for the CEM being 8.5 million. Thus it is relatively lightweight, however, it is more complex compared to other works [4], [30].

The reason for including the two preceding acoustic features from before the emission frame is to use more information about the acoustics. This is done, even though the Conformer Transducer only uses the acoustic features from one time step to produce a prediction during decoding. Moreover, only the preceding frames are included in the CEMs features, not any of the succeeding ones. This is motivated by the findings in [32]. This paper states that the Transducer perform better when using more context and therefore suffers from emission delay since the emission delay is not penalized during training. This kind of penalization is not done in this thesis. The CEM follows the same training scheme as the Conformer Transducer which is influenced by [7].

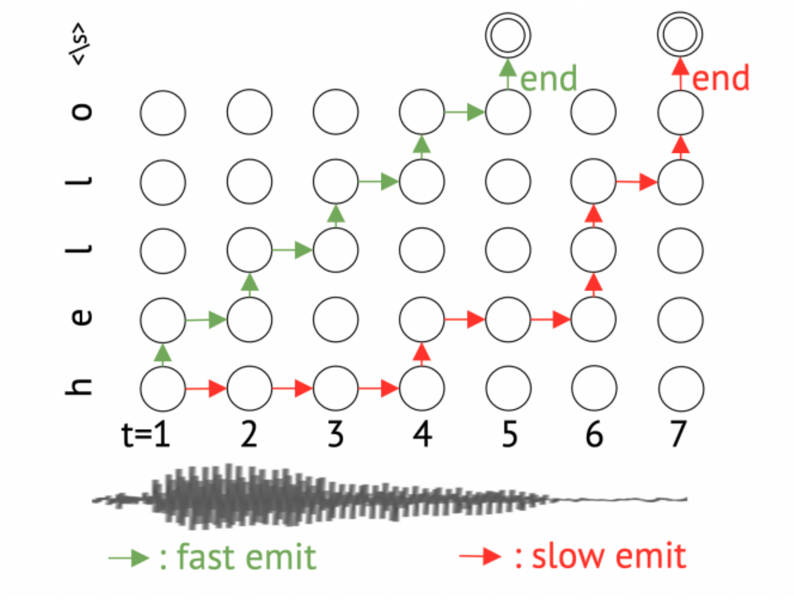


Figure 23: Illustration of delayed emission in the Transducer, taken from [32].

4.3.3 Metrics

In order to measure the performance of the confidence estimations, a metric is necessary. A common metric is the Receiver Operating Characteristic curve (ROC), and additionally the area under this curve. This gives a single value that is easy to compare across models. The ROC is a curve that shows the true positive rate (rate of correct classifications) against the false positive rate (rate of false classifications), for given thresholds given a binary classifier.

As mentioned earlier, the area under the ROC (AUROC) curve is often used as a single measure for the performance of the classifier. If the AUROC is 0.5 the classifier is said to be random. This would be the case if the ROC is a straight line along the diagonal. If the AUROC is equal to one, then the classifier is said to be perfect. This classifier would classify every sample correct at any threshold. At around 0.8 AUROC is considered “good” or sufficiently good classifier. However, what is considered a “good” AUROC is very dependent on the task. For instance, a classifier for roulette with an AUROC of 0.6 would make you a millionaire, but a classifier with the same AUROC for blood type classification would be terrible.

Also, skewed datasets such as in bank fraud detection, the AUROC can be misleading in regards to the actual performance of the classifier. When having a dataset with a lot of easy samples of one class, and only a handful of samples of the other class. This classifier would perform great on these easy samples, but randomly or always wrong on other samples. This would not be reflected in the AUROC.

Another similar metric to the ROC is the Precision-Recall (PR) curve. This is a curve with the precision of the classifier plotted against the recall. The recall is the true positive rate, meaning the fraction of all correctly predicted positive labels, among all positive labels. The precision is the fraction of all the correctly predicted positive labels among all **predicted** positive labels.

$$\begin{aligned} \text{precision} &= \frac{TP}{FP+TP} \\ \text{recall} &= \frac{TP}{FN+TP} \end{aligned} \tag{31}$$

The PR curve and the ROC share the true positive rate as one of the values that make up the metric. However, the PR curve uses the precision in favor of the false negative rate. This results in the PR curve being better for imbalanced datasets. If there is a class imbalance with, for instance, the positive label being a lot less abundant in the set, a random classifier would predict a lot of true samples, even though this is not the case. This leads to the false positives being rate being high compared to the true positives and thus the precision becomes very low.

4.4 Language Model

The confidence scores produced will be used in a downstream task. This is done with a dynamic weighted Beam Search. This Beam Search will rely on a language model that will be incorporated into the Beam Search when testing the system on a test set. The model chosen for this task will have a standard Transformer design. The model will not perform a sequence-to-sequence task such as in for instance language translation. It will instead only take in previous outputs from the Conformer Transducer and output the probabilities for the next token. This behavior does not require the Transformer to attend to a different sequence, which is done in the Transformer decoder, thus only the encoder part of the Transformer is required. The LM consists of 12 encoder layers and has a model dimension of 256 and an output vocabulary with the same size as the Conformer Transducer of $5,000$. This puts the total number of parameters at 12 million. The absolute sinusoidal positional encoding scheme is applied at the input of the model. This positional scheme was chosen since the samples in the dataset used are split into samples of equal length, therefore there is little use for a relative positional encoding scheme. This Transformer encoder will be trained from scratch using the Oscar dataset [33], [34]. This dataset is available in several different languages and consists, as the name suggests, of a lot of text crawled from the internet. This corpus consist of 166 different languages, including Bokmål and Nynorsk. Both the Nynorsk and Bokmål sets were used in training the LM. The two sets were concatenated and then shuffled such that the model would not see just Bokmål samples during the first half of the training and then only Nynorsk during the second half and thus overfitting to one of the two. Little time and effort were used to analyze this dataset as it is quite large and diverse, and it is only used for next token prediction and not something more complex like question answering. The LM follows the same training scheme as the Conformer Transducer and the CEM.

Training a model from scratch was done in favor of using a pretrained model like BERT [35] because it has its own tokenizer, which uses a significantly larger vocabulary than what is used for the Conformer Transducer. This is not an implementation issue, but at a size of $30,000$ rather than $5,000$, this is a significant increase. This will greatly increase the memory consumption during training and result in longer training times due to the lower batch size as a result of the increased memory consumption.

The tokenizer used in this thesis is the SentencePiece [36] tokenizer, which is an unsupervised tokenizer and follows the Byte-Pair-Encoding scheme. This tokenizer is trained on NPSC corpus and the resulting vocabulary is used for both the Conformer Transducer and the LM to make sure the same output node refer to the same token during decoding.

Another consideration is the vocabulary. The BERT model is trained on significantly more text and the tokenizer is designed to fit this dataset. It is possible that the dataset used to train the Conformer Transducer would not contain all the tokens in the BERT tokenizer, thus the Conformer Transducer would have output nodes that is not trained.

The LM will be evaluated on the perplexity score. This is probably the most commonly used metric to report performance for an LM. The perplexity is defined as shown in Equation 32

$$\mathrm{PPL}(X) = \exp\left\{-\frac{1}{t}\sum_{i}^{t}\log p_\theta\left(x_i \mid x_{<i}\right)\right\} \tag{32}$$

This score is defined as the exponential average of the negative log-likelihood of a sequence. The $\log p_\theta(x_i|x_{<i})$ denotes the log-likelihood for the ith output token, conditioned on the tokens leading up to this ith token. Essentially, the exponentiation of the negative log-likelihood. This metric closely resembles the Shannon-entropy, it in fact captures the same information. Thus, given a

perplexity of x, the model would be as confused as if it had to choose between x tokens for its next prediction. An additional thing to keep in mind is that the perplexity is dependent on the vocabulary size of the model, that is, how many output nodes the model has. An untrained LM, with large vocabulary would have a higher and worse perplexity than that of an LM with a smaller vocabulary, since it has more tokens to choose from.

4.5 Dynamic Beam Search

The usability of the confidence scores generated by the CEM is evaluated with a dynamic weighted Language Model Beam Search. The normal, static, way of doing this is by adding the scores from the language model, multiplied by a weight γ, to the scores from the Conformer Transducer for each token.

$$P(y) = P_{ASR}(y) + \gamma P_{LM}(y) \tag{33}$$

When using the CEM the scores from the LM is weighted with a weight γ as previously, but also with $1 - C_{\text{confidence}}$, where $C_{\text{confidence}}$ is the confidence score from the CEM for that token.

$$P(y) = P_{\text{ASR}}(y) + \gamma(1 - C_{\text{confidence}})P_{\text{LM}}(y) \tag{34}$$

The full system is shown in Figure 24. The $\otimes$ blocks denotes multiplication, while $\oplus$ denotes addition. This dynamic extension to the Beam Search is quite simple when all the components of the system are developed, however, the author is not aware of other works using this method.

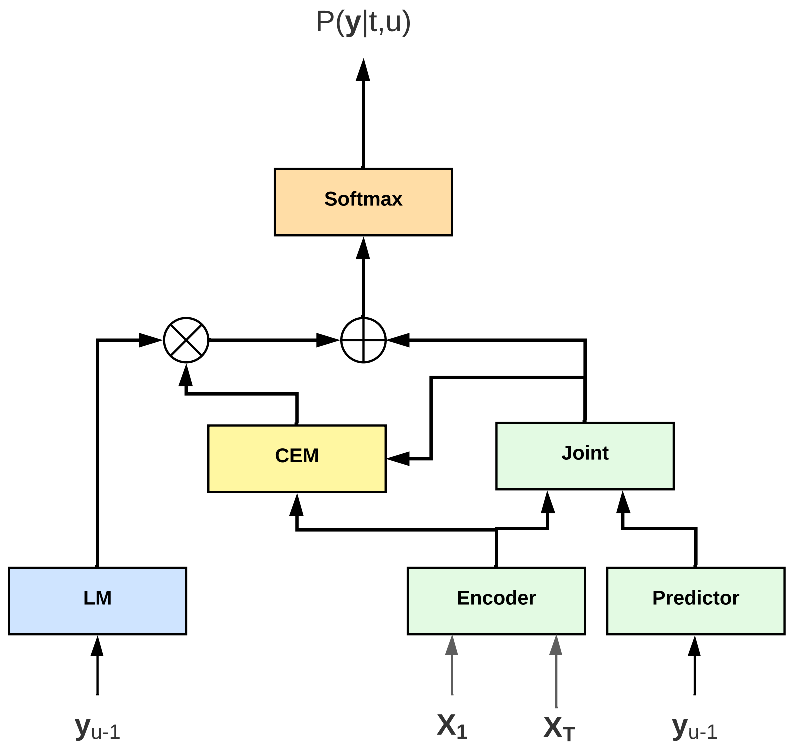


Figure 24: Overview of how all the models developed is tied together in a dynamic Beam Search.

In order to compare the CEM for use in a dynamic Beam Search, its performance will be evaluated against a static Beam Search. All hyperparameters for the Beam Search will be fixed and equal for both the dynamic and static Beam Search, except for the LM weight, γ will be unique to the two methods. The values for the γ will be found with a hyperparameter search. The hyperparameter search will be done using a hyperparameter search library named optuna [37]. The algorithm used for the hyperparameter search will be the Tree-structured Parzen Estimator algorithm. This search will be conducted by choosing a small subset of the validation data to perform the optimization on. This is done to limit the run time needed to find the optimal hyperparameter. The optimal value for both methods will later be used to produce transcriptions on the full test set.

5 Results

5.1 Conformer

The Conformer Transducer has, as mentioned earlier, a WER of 29.41% and a CER 16.22%. However, the performance varies quite a bit from speaker to speaker and depending on the sentence being spoken. The model struggles a lot with the names of people, some examples of this are shown below.

> **Ground Truth:** neste taler er **Solfrid Lerbrekk**
>
> **Prediction:** neste taler er **Sandra Borch**

> **Ground Truth:** neste taler er **Ingvild Kjerkol**
>
> **Prediction:** neste taler er **Liv Signe Navarsete**

This is likely the case of the prediction network in the Conformer Transducer "winning" over the acoustic encoder, and thus outputting something the predictor has seen many times during training. These types of errors are almost impossible to correct with a language model during a Beam Search since the prior tokens give no information of the name of the speaker.

An example of another type of error is shown below.

> **Ground Truth:** det kunne **gått** mye bedre om det **bare var** en annen regjering
>
> **Prediction:** det kunne **godt** mye bedre om det **var bare** en annen regjering

In this case, one of the errors is the confusion of two words with quite equal vocal pronunciation. This is an error that a LM could correct. However, it might not be so easy for the LM to detect this kind of error in a Beam Search since the LM does not have information about the succeeding words after the erroneous word. These are the words that actually make the correct word much more likely. This kind of error would be easier to detect with a rescoring algorithm that has access to the whole sentence and thus more context for each word.

An example of a sentence that could in theory be easier for an LM in a Beam Search to correct is shown below. This sentence has an error in a common phrase or saying in Norwegian. The preceding text makes it such that predicting the phrase is not so hard.

> **Ground Truth:** regjeringa beskriver verden med solskinn som skinner over **gull og grønne skoger** og alt går godt
>
> **Prediction:** regjeringa beskriver verden med solskinn som skinner over **gul og gründer skog** og alt går godt

5.2 CEM

The CEM is tested on the same test set as the Conformer Transducer. The overall performance of the CEM is decent but not on the same level as the Confidence Estimation Modules in [30]. The AUROC on the test set is 0.83 and the AUC PR is 0.66.

The ROC and PR curves are plotted in Figure 25 and Figure 26 respectively.

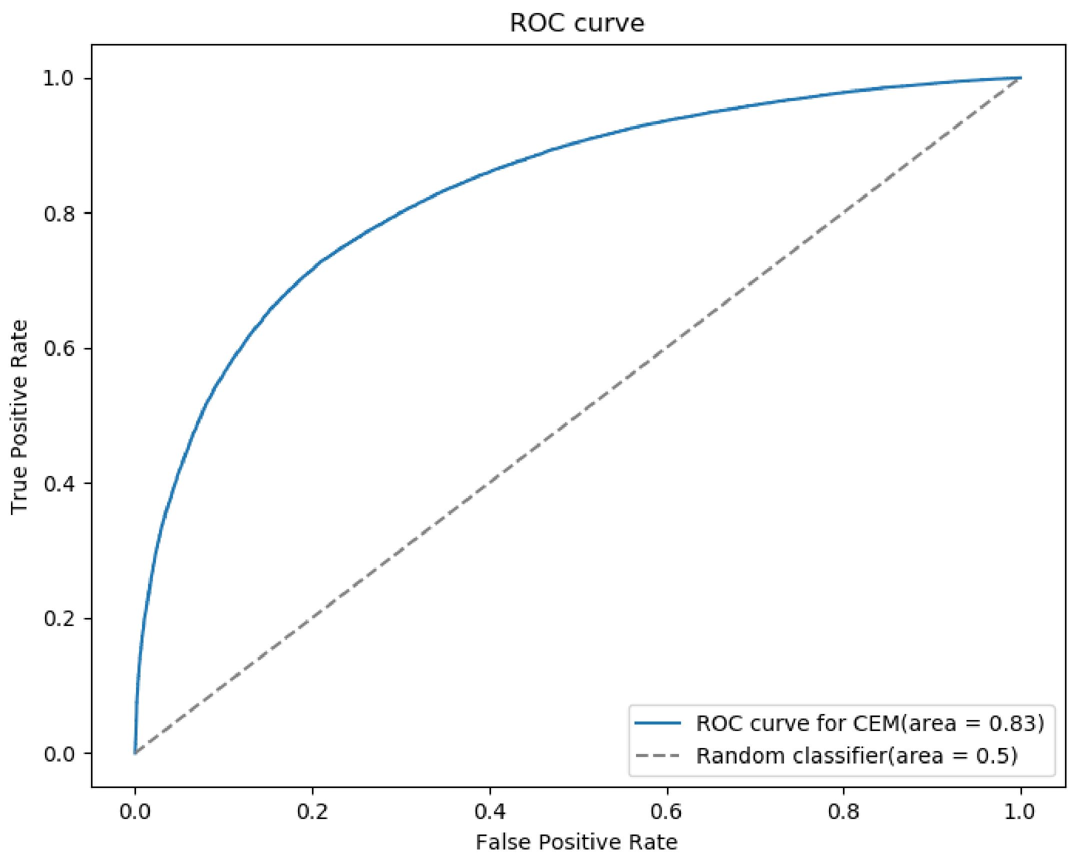


Figure 25: ROC for the Confidence Estimation Module from the test set. The dotted line on the diagonal represents a random classifier

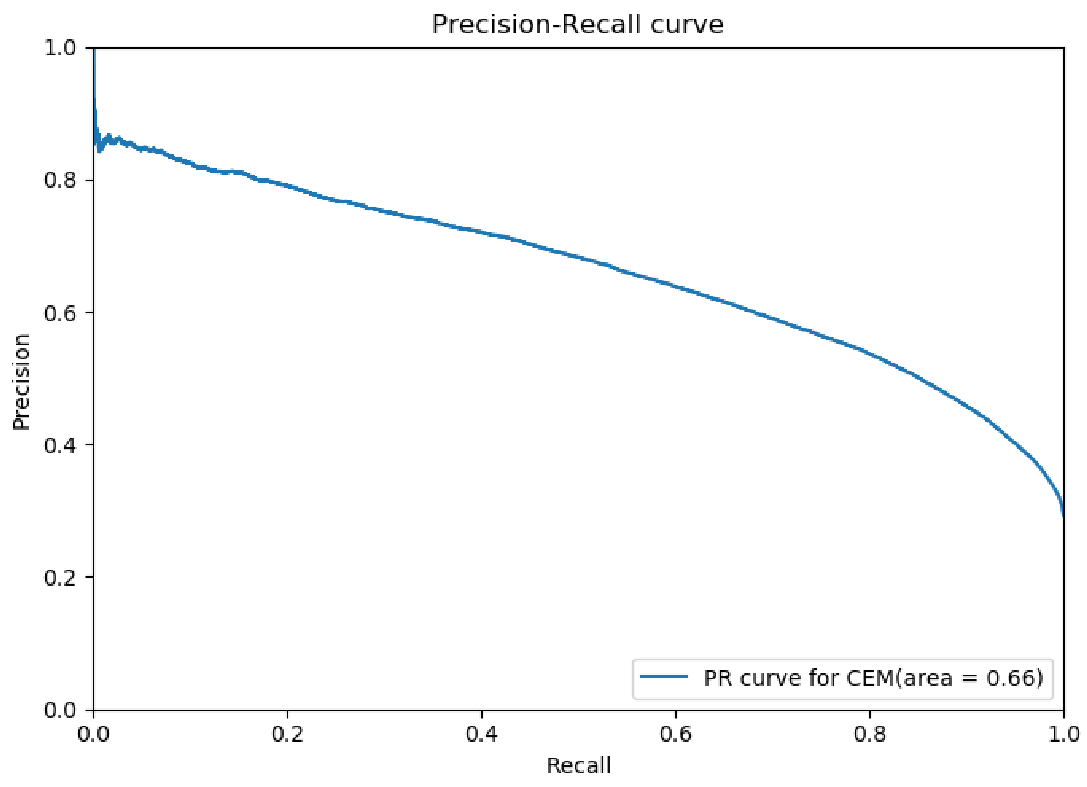


Figure 26: PR curve for the Confidence Estimation Module from the test set

The class used as "Positive" to calculate the precision and recall is set to be the class with the least amount of samples. In this case, this is the incorrectly predicted words. This is only done for the purpose of correctly calculating the PR curve.

The number of positive samples in the test set, meaning tokens that are part of a correct word is 76898. The number of negative samples in the test set, meaning tokens that are part of a wrong transcribed word is 31610. This confrms that the dataset is indeed unbalanced.

5.3 LM

It is quite hard to evaluate the performance of the LM, other than using a perplexity metric. Some LMs are trained at question answering, translation, or summary generation. The model developed for the dynamic Beam Search is however trained at predicting the next token. Predicting the next correct token is quite a challenging task, even for humans. The perplexity of the LM on the OSCAR test set is 27. The LM has a vocabulary size of $5,000$, the same as the Conformer Transducer.

5.4 Dynamic Beam Search

A total of 25 hyperparameter searches to find the best performing LM weight, γ, was done on a small subset of the validation set, 75 samples. A subset was used instead of the full set since the full validation set with beam search is quite time-consuming, taking nearly 9 hours. This subset was picked randomly. The best result from the static Beam Search had a WER of 25.78%, while the best result from the dynamic Beam Search had a WER of 24.78%.

The dynamic Beam Searches are less prone to suffer from incorrect γ. Of the top ten performing runs, all of them are from dynamic Beam Searches. The best static Beam Search ranks in number eleven. The γ used in the top ten dynamic Beam Searches range from 0.16 to 0.33 and the WER range from 24.78% to 25.54%. For the static Beam Search the γ is a more sensitive parameter. This is illustrated in Figure 27

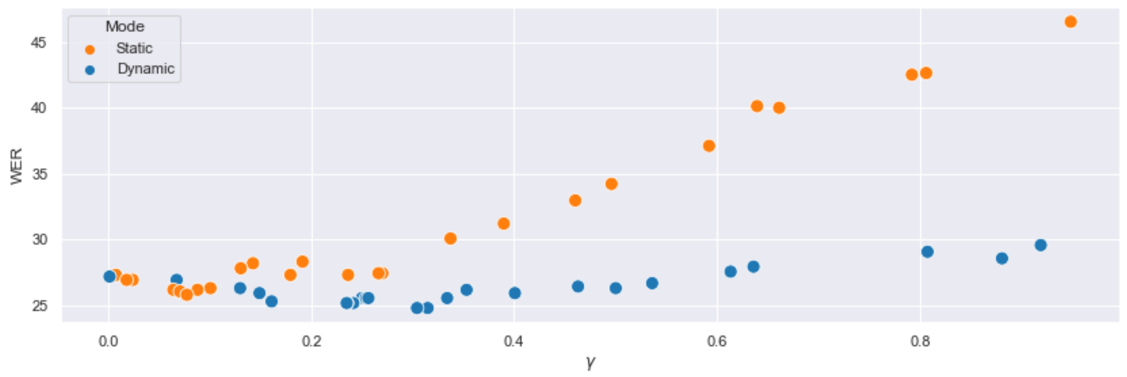


Figure 27: Different values of the weight γ plotted against the resulting WER for both dynamic and static Beam Search.

The best-performing parameters for the weight γ were used to test the performance of the Beam Searches on the full test set. The results are shown in Figure 28, Figure 29, Figure 30 and Figure 31. The total WER for the dynamic Beam Search was 28.5% while the WER for the static Beam Search was 27.9%. Thus, the static Beam Search performed better when using the full test set. The performance difference between the two methods was quite consistent, however, the CER for Nynorsk samples was significantly lower for the dynamic Beam Search, at 24.4% compared to 40%.

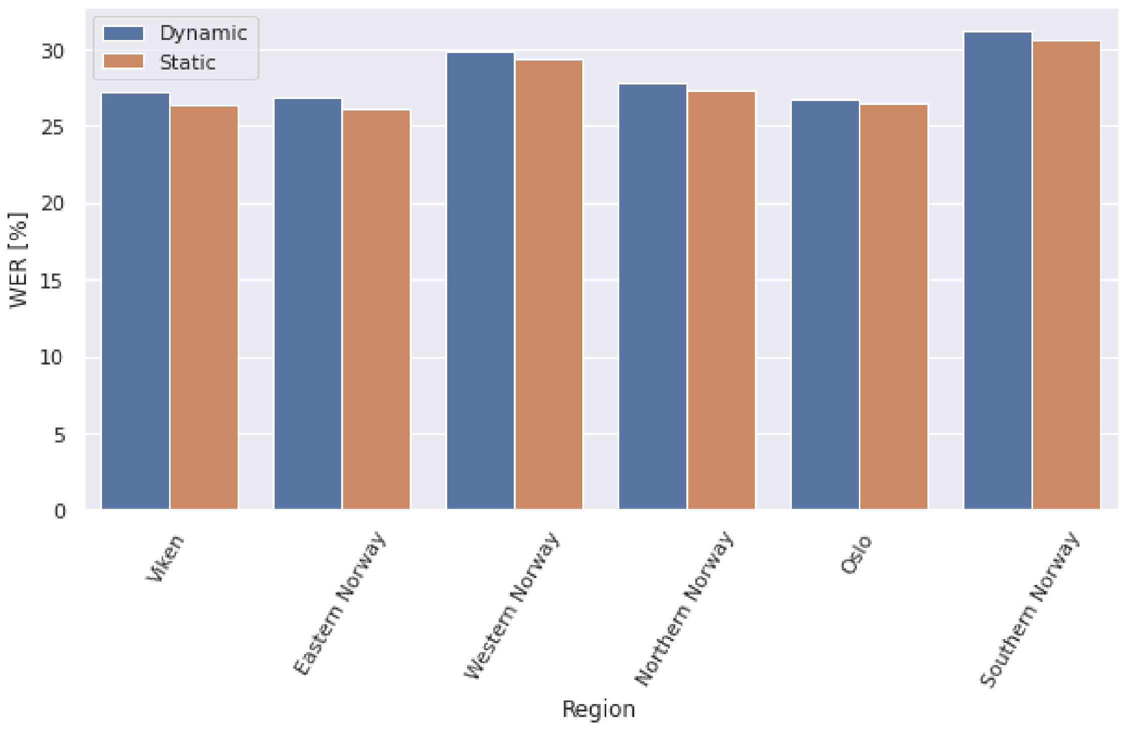


Figure 28: The WER, y-axis, for each region, x-axis, in the test set. Blue is for dynamic Beam Search with optimized γ, orange is for static Beam Search with optimized γ.

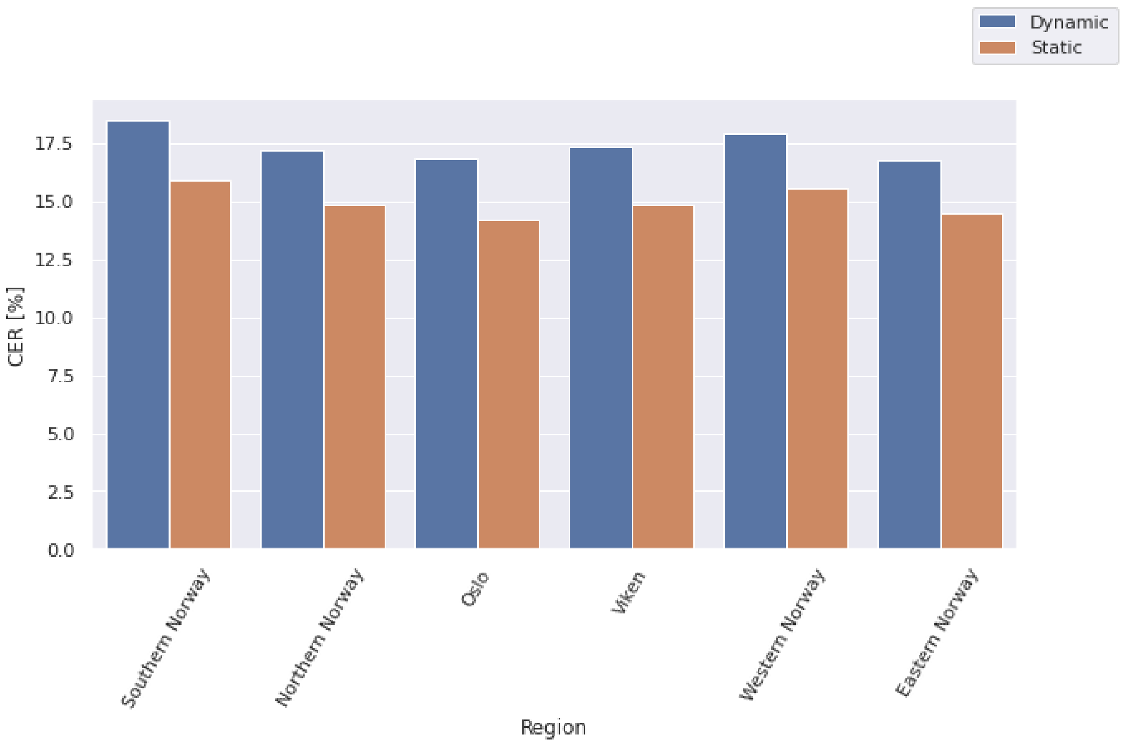


Figure 29: The WER, y-axis, for each region, x-axis, in the test set. Blue is for dynamic Beam Search with optimized γ, orange is for static Beam Search with optimized γ.

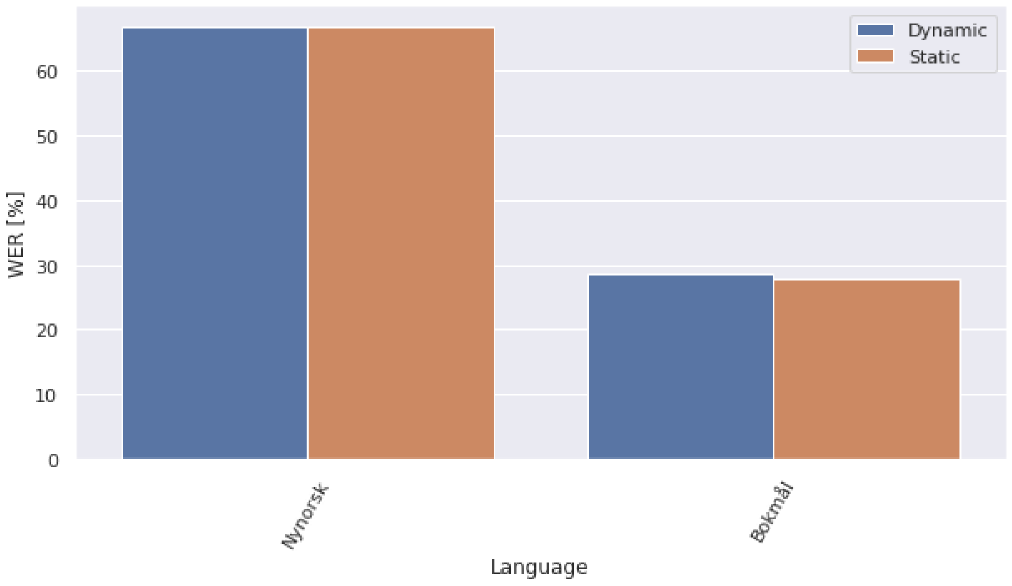


Figure 30: Results from the full Beam Search test. The WER is on the y-axis, with the written language on the x-axis. Blue is for dynamic Beam Search with optimized γ, orange is for static Beam Search with optimized γ.

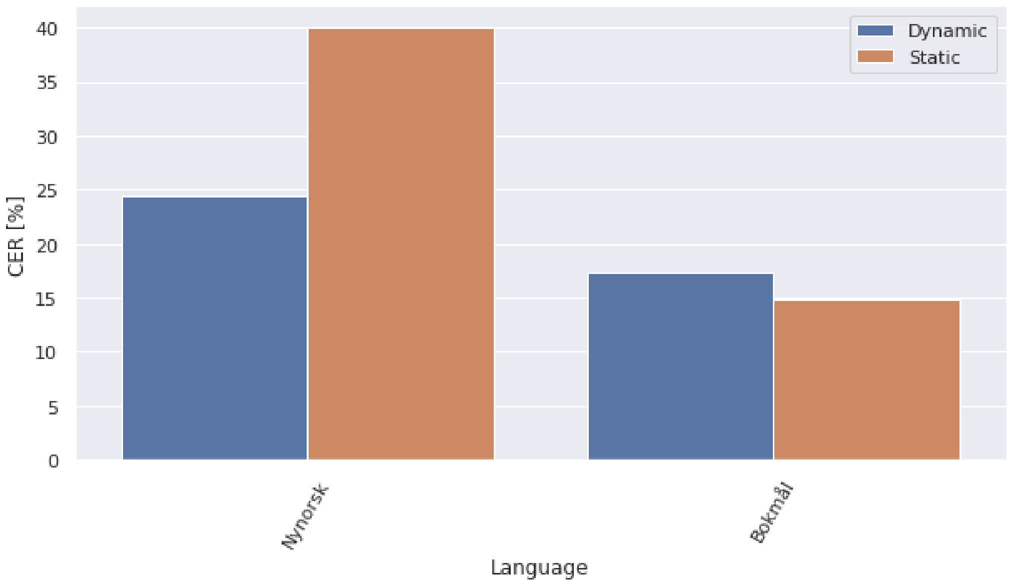


Figure 31: Results from the full Beam Search test. The CER is on the y-axis, with the written language on the x-axis. Blue is for dynamic Beam Search with optimized γ, orange is for static Beam Search with optimized γ.

6 Discussion

Conformer

The Conformer Transducer has subpar performance compared to its reported capabilities [18]. This is likely due to the amount of data it has been trained on. The original Conformer Transducer and other architectures with similar performance have been trained and tested on the English dataset LibriSpeech 960 dataset, which consists of 960 hours of transcribed speech. In other words, significantly more data than what is used in this thesis. The level of difficulty of the dataset also plays a significant part in the reported performance of the Conformer Transducer. Spontaneous speech is significantly harder to transcribe than planned speech, the samples in the NPSC dataset are a mix of the two. At times, the sentences are coherent and follow a conventional structure, but at times, people have hesitations, and sometimes use wrong words. The dataset also contains two different written languages, Nynorsk and Bokmål, and a wide variety of dialects which all contribute to making the task of transcription harder. There are several methods to increase the performance of the Conformer Transducer. Previous work has shown that the performance may be increased by adding more data from another dataset or even pre-training with a dataset in another language like the LibriSpeech 960. Contrastive pre-training of the acoustic encoder is also an alternative to increasing performance, but this has a significant computational cost [5]. This could all have been done for the sake of improving the performance of the Conformer Transducer, however, the performance of the Conformer Transducer is not the main objective of this thesis, thus there has been little effort directed towards this task. However, an argument for a better performing Conformer Transducer is that the CEM might also benefit from a better Conformer Transducer, like in [30]. The theory is that a better acoustic encoder would be better to distinguish between words and that it would be easier to identify words with low confidence since these words are hard to transcribe even for humans.

Confidence Estimation Module

The CEM is also not quite performing as reported in other works [4], [30]. With an AUROC of 0.83 and an AUC PR of 0.66, the CEM developed for this thesis has quite a substantial performance deficit compared to the reported performance in [30] which is 0.99 for both of the metrics. The ASR system used in [30] has a reported performance advantage compared to Conformer Transducer, with a WER of 21.1% on a conversational set and 10.8% WER on a dictation set. The NPSC would be a dataset closer to the dictation set, thus the performance difference is significant. As mentioned earlier, it might be the case that the CEM would benefit from a better-performing ASR system.

The labels for each token are created by evaluating if the whole predicted word is correct. This results in tokens which are correct are labeled as wrong, as in [30]. In the dynamic Beam Search the token level confidence scores from the CEM is used, while the CEM is trained to predict the confidence for tokens on a word level. Thus, the CEM might benefit from being trained with the labels being generated on a token level when used in a dynamic Beam Search.

Language Model

The LM is a standard Transformer architecture, and even though it is quite powerful, it is no longer the state of the art. The LM is also quite small and has a small vocabulary making it easy to achieve a good perplexity score. The dataset might not be the best dataset to train this LM, since the LM is intended to be used on speech rather than text. There might be some differences between the way the speakers would say a sentence in contrast to a sentence that is written in a text. The speakers in the dataset would likely have a written manuscript and thus do not perform spontaneous speech, nevertheless, there is a difference between this and the text in the corpus which is crawled from the net. A larger and more advanced LM would likely be better also, but this is also a time-consuming job. Either to train a new large model like BERT, or train a new head from a pre-trained large model. As a proof of concept, the LM is more than adequate and efforts to improve the system as a whole is better spent if focused on the Conformer Transducer and CEM.

Dynamic Beam Search

The dynamic Beam Search shows signs of promising results when tuning the hyperparameter γ, with great performance compared to static Beam Search. From Figure 27, the dynamic Beam Search seems to be less sensitive when it comes to values of the weight γ. Thus making it easier and less time-consuming to tune this parameter. The confidence scores from the CEM look like it manages to cancel out the usage of the LM for words it is very confident for. This is a very promising result that can prove that CEM is usable in a dynamic Beam Search. When using the full test set for decoding, the story becomes different however, the static Beam Search quite consistently performs the best. This is a bit of a surprise considering the results found during the tuning. A possible explanation is that the weight γ is overfitted on this smaller set of data. This is a contradiction to the statement that the dynamic Beam Search is less sensitive to sub-optimal values of γ. It is also important to keep in mind that the performance of the ASR system is not the best. Especially for Nynorsk, the model performs poorly judging from the WER, it may, however, produce meaningful transcriptions which are mainly in Bokmål, since this is the written language it has seen the most.

It is quite challenging to know what samples are supposed to be transcribed in Nynorsk and what samples are supposed to be transcribed as Bokmål, when a person is using a dialect. This is not a standard and it is unclear what types of guidelines the makers of the dataset has followed when producing the ground truth labels.

The performance difference of the Beam Search with an LM compared to without an LM is quite comparable to the experiments done in [16]. The WER reduction is 3.09% and 5.13% for the dynamic and static Beam Search respectively, compared to a static Beam Search without an LM. While not negligible, does not make the system perform at the level of other ASR systems trained with more data [18]. The original Conformer also used an LM during decoding and had quite substantial relative performance gains when using an LM, 13.00% and 14.00% reduction in WER was achieved on the two test sets used [18]. This confirms that the use of an LM during decoding is a great way of increasing performance despite the increase in parameters. The main difference between these two systems is the amount of training data that is used. Thus, as a result, the system with more training data performs better. It would be interesting to see if a dynamic Beam Search would increase performance for a better-performing ASR system. Not only because the errors this system produces are different than the errors produced by the ASR used in this thesis, but also testing the hypothesis that the CEM would also perform better. From the visual inspection of the errors, it is clear that some of the errors are quite challenging for an LM to correct, names are for instance quite challenging both for the ASR and the LM.

7 Conclusion

This thesis has had a focus on ASR and confidence estimation related to the transcriptions of speech samples. A model for confidence estimation has been developed and trained to work with the ASR system used in this thesis. The CEM developed shows decent performance with an AUROC of 0.83 and an AUC PR of 0.66. Despite this, it does not match state-of-the-art results within confidence estimation for ASR. This lack of performance is likely due to the lack of performance of the ASR system since it forms the basis of the training data used for the CEM. Also, a better-performing acoustic encoder would be better at distinguishing words and sounds from another.

The CEM was later used in a dynamic Beam Search and tested against a normal static Beam Search, both Beam Searches were done with an added LM. This LM was trained on the OSCAR dataset and has a perplexity of 27. This LM shares the same vocabulary and number of output nodes as the ASR system in order to be compatible with the Beam Search. The dynamic Beam Search initially shows promising results, with it being easier to tune the hyperparameter γ, and also showing better and more consistent results on a smaller subset of the test data for a wide range of values of γ. On the full test set, the normal static Beam Search performs better, this is hard to explain. The γ used for the test being overfitted on the smaller test set, is a likely theory. However, this somewhat invalidates the claim that the γ for the dynamic Beam Search with the CEM is less sensitive and therefore easier to tune.

7.1 Future Work

Given the performance of the CEM developed in this thesis compared to other works [30], there is definitely room for more improvements when it comes to performance. The performance of the CEM is likely heavily linked to the performance of the ASR system. This performance can be increased in several different ways, without changing the architecture of the model. These methods include pre-training and using more high quality data. The size of the model can likely also be reduce with better data from a better-performing ASR. This will reduce the computing cost and system requirements for incorporating the CEM into another system. The CEM might be useful for eliminating the use of the LM during decoding for high confidence words. This might have a great impact on the run time of the system, since fewer forward passes of the LM is required. Using the CEM in other tasks and applications, rather than just confidence estimation is an interesting topic. Its usage in other tasks, such as unsupervised learning with a Teacher-Student configuration is possible and should be looked into to see if it can result in smarter use of unsupervised data.

Bibliography

[1] T. B. Brown, B. Mann, N. Ryder *et al.*, *Language models are few-shot learners*, 2020. DOI: 10.48550/ARXIV.2005.14165. [Online]. Available: https://arxiv.org/abs/2005.14165.

[2] C. Raffel, N. Shazeer, A. Roberts *et al.*, *Exploring the limits of transfer learning with a unified text-to-text transformer*, 2019. DOI: 10.48550/ARXIV.1910.10683. [Online]. Available: https://arxiv.org/abs/1910.10683.

[3] W. Fedus, B. Zoph and N. Shazeer, *Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity*, 2021. DOI: 10.48550/ARXIV.2101.03961. [Online]. Available: https://arxiv.org/abs/2101.03961.

[4] Q. Li, D. Qiu, Y. Zhang *et al.*, 'Confidence estimation for attention-based sequence-to-sequence models for speech recognition', *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6388–6392, 2021.

[5] S. Schneider, A. Baevski, R. Collobert and M. Auli, *Wav2vec: Unsupervised pre-training for speech recognition*, 2019. DOI: 10.48550/ARXIV.1904.05862. [Online]. Available: https://arxiv.org/abs/1904.05862.

[6] K. Stensgård, *Automatic speech recognition for norwegian with attention-based models*, Project report in TFE4580, Dec. 2021.

[7] A. Vaswani, N. Shazeer, N. Parmar *et al.*, *Attention is all you need*, 2017. DOI: 10.48550/ARXIV.1706.03762. [Online]. Available: https://arxiv.org/abs/1706.03762.

[8] A. Dosovitskiy, L. Beyer, A. Kolesnikov *et al.*, *An image is worth 16x16 words: Transformers for image recognition at scale*, 2020. DOI: 10.48550/ARXIV.2010.11929. [Online]. Available: https://arxiv.org/abs/2010.11929.

[9] H. M. Fayek, *Speech processing for machine learning: Filter banks, mel-frequency cepstral coefficients (mfccs) and what's in-between*, 2016. [Online]. Available: https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html.

[10] S. S. Stevens, J. Volkmann and E. B. Newman, 'A scale for the measurement of the psychological magnitude pitch', *The Journal of the Acoustical Society of America*, vol. 8, no. 3, pp. 185–190, 1937. DOI: 10.1121/1.1915893. eprint: https://doi.org/10.1121/1.1915893. [Online]. Available: https://doi.org/10.1121/1.1915893.

[11] D. S. Park, W. Chan, Y. Zhang *et al.*, 'Specaugment: A simple data augmentation method for automatic speech recognition', *Interspeech 2019*, Sep. 2019. DOI: 10.21437/interspeech.2019-2680. [Online]. Available: http://dx.doi.org/10.21437/Interspeech.2019-2680.

[12] A. Graves, *Sequence transduction with recurrent neural networks*, 2012. arXiv: 1211.3711 [cs.NE].

[13] L. Lugosch, *Sequence-to-sequence learning with transducers*, Nov. 2020. [Online]. Available: https://lorenlugosch.github.io/posts/2020/11/transducer/.

[14] D. Bahdanau, K. Cho and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, 2016. arXiv: 1409.0473 [cs.CL].

[15] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei and I. Sutskever, 'Language models are unsupervised multitask learners', 2019.

[16] R. Cabrera, X. Liu, M. Ghodsi, Z. Matteson, E. Weinstein and A. Kannan, *Language model fusion for streaming end to end speech recognition*, 2021. DOI: 10.48550/ARXIV.2104.04487. [Online]. Available: https://arxiv.org/abs/2104.04487.

[17] W. Chan, N. Jaitly, Q. V. Le and O. Vinyals, *Listen, attend and spell*, 2015. arXiv: 1508.01211 [cs.CL].

[18] A. Gulati, J. Qin, C.-C. Chiu *et al.*, *Conformer: Convolution-augmented transformer for speech recognition*, 2020. arXiv: 2005.08100 [eess.AS].

[19] M.-T. Luong, H. Pham and C. D. Manning, *Effective approaches to attention-based neural machine translation*, 2015. arXiv: 1508.04025 [cs.CL].

[20] Q. Zhang, H. Lu, H. Sak *et al.*, *Transformer transducer: A streamable speech recognition model with transformer encoders and rnn-t loss*, 2020. arXiv: 2002.02562 [eess.AS].

[21] K. Cho, B. van Merrienboer, C. Gulcehre *et al.*, *Learning phrase representations using rnn encoder-decoder for statistical machine translation*, 2014. arXiv: 1406.1078 [cs.CL].

[22] Y. Lu, Z. Li, D. He *et al.*, *Understanding and improving transformer from a multi-particle dynamic system point of view*, 2019. arXiv: 1906.02762 [cs.LG].

[23] P. Ramachandran, B. Zoph and Q. V. Le, *Searching for activation functions*, 2017. arXiv: 1710.05941 [cs.NE].

[24] J. L. Ba, J. R. Kiros and G. E. Hinton, *Layer normalization*, 2016. arXiv: 1607.06450 [stat.ML].

[25] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le and R. Salakhutdinov, *Transformer-xl: Attentive language models beyond a fixed-length context*, 2019. DOI: 10.48550/ARXIV.1901.02860. [Online]. Available: https://arxiv.org/abs/1901.02860.

[26] Y. Gal, 'Uncertainty in deep learning', Ph.D. dissertation, University of Cambridge, 2016.

[27] B. Lakshminarayanan, A. Pritzel and C. Blundell, *Simple and scalable predictive uncertainty estimation using deep ensembles*, 2016. DOI: 10.48550/ARXIV.1612.01474. [Online]. Available: https://arxiv.org/abs/1612.01474.

[28] S. Fort, H. Hu and B. Lakshminarayanan, *Deep ensembles: A loss landscape perspective*, 2019. DOI: 10.48550/ARXIV.1912.02757. [Online]. Available: https://arxiv.org/abs/1912.02757.

[29] T. Papamarkou, J. Hinkle, M. T. Young and D. Womble, *Challenges in markov chain monte carlo for bayesian neural networks*, 2019. DOI: 10.48550/ARXIV.1910.06539. [Online]. Available: https://arxiv.org/abs/1910.06539.

[30] M. Wang, H. Soltau, L. E. Shafey and I. Shafran, 'Word-level confidence estimation for rnn transducers', 2021. DOI: 10.48550/ARXIV.2110.15222. [Online]. Available: https://arxiv.org/abs/2110.15222.

[31] P. E. Solberg and P. Ortiz, *The norwegian parliamentary speech corpus*, 2022. DOI: 10.48550/ARXIV.2201.10881. [Online]. Available: https://arxiv.org/abs/2201.10881.

[32] J. Yu, C.-C. Chiu, B. Li *et al.*, *Fastemit: Low-latency streaming asr with sequence-level emission regularization*, 2020. DOI: 10.48550/ARXIV.2010.11148. [Online]. Available: https://arxiv.org/abs/2010.11148.

[33] P. J. Ortiz Su'arez, L. Romary and B. Sagot, 'A monolingual approach to contextualized word embeddings for mid-resource languages', in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Online: Association for Computational Linguistics, Jul. 2020, pp. 1703–1714. [Online]. Available: https://www.aclweb.org/anthology/2020.acl-main.156.

[34] P. J. Ortiz Su'arez, B. Sagot and L. Romary, 'Asynchronous pipelines for processing huge corpora on medium to low resource infrastructures', en, P. Bański, A. Barbaresi, H. Biber *et al.*, Eds., ser. Proceedings of the Workshop on Challenges in the Management of Large Corpora (CMLC-7) 2019. Cardiff, 22nd July 2019, Mannheim: Leibniz-Institut f"ur Deutsche Sprache, 2019, pp. 9–16. DOI: 10.14618/ids-pub-9021. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:bsz:mh39-90215.

[35] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2018. DOI: 10.48550/ARXIV.1810.04805. [Online]. Available: https://arxiv.org/abs/1810.04805.

[36] T. Kudo and J. Richardson, *Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing*, 2018. arXiv: 1808.06226 [cs.CL].

[37] T. Akiba, S. Sano, T. Yanase, T. Ohta and M. Koyama, *Optuna: A next-generation hyperparameter optimization framework*, 2019. DOI: 10.48550/ARXIV.1907.10902. [Online]. Available: https://arxiv.org/abs/1907.10902.