Karlsruhe Institute of Technology

Online Neural Network-based Language Identification

Master's Thesis of

Daniel H. Draper

at the Department of Informatics
Institute for Anthropomatics and Robotics

Reviewer:	Prof. Dr. Alexander Waibel
Second reviewer:	
Advisor:	M.Sc. Markus Müller
Second advisor:	Dr. Sebastian Stüker

12. December 2016 – 11. May 2017

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, 12th of May, 2017

. . . . . .

(Daniel H. Draper)

Abstract

Today, speech recognition has become ubiquitous in human machine interfaces (HMIs). Be it voice control, speech translation or personal assistants, the speech recognition component always relies on the source language being known beforehand. This is due to the fact, that single-language speech recognizers perform better than multi-lingually trained ones. Language Identification (LID) could therefore improve usability and performance of speech-based technology, without requiring any human interaction of selecting the correct source language. Online performance, as well as speed in these HMIs is of utmost importance, which is why we evaluate all our approaches in consideration of this aspect.

This thesis investigates a neural-network based approach to LID. Current approaches to language identification are presented and compared. The results in this thesis compare different network structures, different audio preprocessing and network-post-processing. Three different data corpora are used to verify results with different data sets. The best neural network structure gives a relative improvement of 18% over our baseline setup.

We also evaluate different post-processing approaches to further improve classification results. We introduce our own metric of the "Out-of-Language-Error" (OLE) that measures the "noisiness" of the filter/net output. Overall, a basic counting filter, a maximum-sequence filter and a gauss smoothing produces the best results and could increase performance in an online implementation.

Zusammenfassung

Heutzutage ist Automatische Spracherkennung (ASR) aus der Mensch-Maschinen-Kommunikation (MMK) nicht mehr wegzudenken. Ob es Sprachkontrolle, Sprachübersetzung oder persönliche Assistenten auf dem Smartphone sind, der ASR Teil beruht darauf, dass die Eingangssprache vor der Erkennung bekannt ist. Dies verbessert die Leistung des Spracherkenners, da auf einer Sprache trainierte Spracherkenner eine geringere Fehlerrate haben als multilingual-trainierte. Daher kann die Sprachenidentifizierung (LID) in dieser MMK die Benutzerfreundlichkeit und Leistung verbessern, da keine Aktion des Benutzers mehr erforderlich ist. Da diese MMK in der heutigen Technologie immer auf Online-Performanz und Geschwindigkeit bedacht ist, evaluieren wir alle unsere Ansätze unter diesem Gesichtspunkt.

Diese Arbeit untersucht einen Ansatz der LID, der Neuronale Netze benutzt. Auch stellen wir heutige Ansätze der LID der Literatur dar und vergleichen sie mit Unserem. In der Arbeit haben wir verschieden Netzwerkstrukturen, verschiedene Audio-Vorverarbeitungen und Netzwerk-Nachverarbeitungen evaluiert. Drei verschiedene Datensätze werden eingeführt auf denen wir alle unsere Ergebnisse testen. Die beste Netzstruktur resultierte in einer relativen Verbesserung von 18 % gegenüber unserem ursprünglichen Aufbau.

Wir haben auch verschiedene Nachverarbeitungen der Neuronalen Netz-Ausgabe verglichen. Hierfür führen wir unser eigene Metrik den “Out-of-Language-Error” (OLE) ein, der das “Rauschen” der einzelnen Filter/des Netzes misst. Insgesamt produzieren ein einfacher Zählfilter, ein maximal-Sequenz Filter und ein Gauß-Filter die besten Ergebnisse und könnten die Ergebnisse eines LID Netzes in einer konkreten Implementierung stark verbessern.

Contents

List of Figures

List of Tables

1. Introduction

Language Identification (LID) describes the classification task of differentiating between spoken speech in different languages and being able to correctly classify which speech-segments consists of which language. Neural Networks refer to Artificial Neural Networks (ANNs), a Machine Learning approach to classification tasks. They are employed greatly throughout all sciences and especially in computer science and tasks concerned with the processing of speech. This thesis proposes a low-latency, fast, and "online", approach to Language Identification using ANNs.

The work done in this thesis uses the Janus Recognition Toolkit (JRTk) [**?**], an Automatic Speech Recognition toolkit developed in joint cooperation by the KIT and CMU. The JRTk offers a tcl/tk[1] script-based environment for the development of Automatic Speech Recognition (ASR) systems, therefore source code in this thesis will consist of tcl/tk scripts with (some) Janus-specific commands. The JRTk and tcl/tk are further explained in sec. 3.2.

1.1. Motivation

ASR is used in many applications and devices today, especially in the rise of handheld mobile devices like smartphones and tablets. It has progressed quickly in the last five years and has found commercial success. Famous examples include Google's[2] "Ok, Google" and Apple's[3] Siri. Which both include voice search[FHBM08] and a form of voice control, that even is extensible in the case of Google and Android e.g[bAO14]. Many other applications have emerged, including spoken language translation[4], especially relevant for this thesis in the realm of spoken language translation, in the form of the Lecture Translator[MNN+16]
.

The task of LID can be applied in all of those fields, as the best-performing ASR is trained on one language and therefore requires changing the trained model when the input language changes. This requires user interaction in the form of defining the input language. Robust and low-latency language identification could eliminate the need for this interaction and make the user experience more streamlined.

LID would be especially applicable in the realm of spoken language translation, as used for example in the European Parliament where already components of ASR and Machine

[1] Tcl/tk: `https://www.tcl.tk/`
[2] Google: `www.google.com`
[3] Apple: `www.apple.com`
[4] IWSLT: `iwslt.org`

Translation are employed and are being actively developed in the TC-STAR initiative[5], e.g as in[VMH+05], and LID would further be able to automate these translation tasks.

This thesis will focus mostly on the KIT's Lecture Translator[6] as the system trained was implemented for it. We believe our results are generic enough to be transferable to other applications: e.g. fully-automated European Parliament speech translations, or with small implementation-specific changes.

1.2. Overview

This thesis is set up as follows: Chapter 2 introduces related and previous work in the realm of Language Identification. Chapter 3 gives an introductory view of LID, and further defines the task this thesis tries to solve. Afterwards we give preliminary theoretical explanations and definitions, including an introduction to Neural Networks in Sec. 3.3. Chapter 4 describes the experimental setup used in this thesis, including the data corpora. Audio Preprocessing is explained in detail in chapter 5.

Chapters 6 describes our results that were accomplished by evaluation of various network setups and layouts, e.g. the amount of hidden layers and neurons. Different smoothing mechanisms on top of the direct neuronal output layer of the network are explained in chapter 7. This is followed by the final summary of our work and an outlook on future work in chapter 8.

[5]TC-STAR: `tcstar.org`

[6]Lecture Translator: `https://lecture-translator.kit.edu`

2. Related Work

This chapter introduces related and previous works in the realm of Language Identification (LID) and how those approaches differ from the ones employed in the following chapters.

2.1. Historically

As presented in [Z+96] by Zissman, Language Identification has historically been the most successful using single-language phone recognition followed by language-dependent, interpolated n-gram language modeling (PRLM) or multiple single-language phone recognizers and language-dependent parallel phone recognition (PPRM). Zissman evaluates PRLM and PPRM as well as the worse performing Gaussian Mixture Models (GMM) on the Oregon Graduate Institute Multi-Language Telephone Speech Corpus (OGI) [MCO92]. In his results on the relatively limited speech corpus of only 90 minutes per language, the PPRM perform the best with an 2-Language average error of 8 % and the GMM falling down to an error of 23 %.

In [TCSK+02], Torres-Carrasqiullo et al. present two GMM-based approaches to LID that use shifted-delta-cepstral (SDC) coefficients as the feature input vectors to achieve comparable LID (and computationally much faster) results. In opposition to PRLM and PPRM, GMM use the acoustic content of the speech signal to classify languages. The GMM with SDC inputs fare the same as PPRM on the CallFriend Corpus [Con96]. The same OGI test set as [Z+96] for 10-second utterances is only about 9 % worse for GMM. However they start being greatly outperformed by PPRM in longer 45 second utterances. GMM do have a significant lesser computational overhead [Z+96] [TCSK+02], and do not require *a priori* information (phonetic transcriptions) to be available [ZB01], [Z+96] making their research still worthwhile.

2.2. Recent Approaches

In [LML07] Li et al. used vector space modeling (VSM) in combination with the already presented PPRM to calculate the distance between different languages more easily thus making classification easier. They determine the top most occurring words in a language, then rely on these statistics to distance languages from each other. After they apply a language-dependent language model on top for classification. A similar approach is proven to be successful with support vector machines (SVM) in [DL08], and further substantiated

in [ML06] where the SVM approach performs the best out of a comparison with the older PPRM and VSM methods.

Recently with the advance of neural networks in machine learning, many different approaches have been tried. In Leena et al. [LRY05] present a Feed-Forward Neural Network that is trained using both phonotactic (syllable occurrence, co-occurrence of syllables, unique syllables and pronunciation variations) and prosodic (rhythm and intonation of speech) features, giving a feature vector of 33 dimensions, to recognize the language. Leena et al. also use autoassociative neural networks and spectral similarity approaches in [MY04] to recognize language reasonably well using only a few seconds of the speech as input.

I-vector approaches as in [SJB+13], [DEGP+12] have also recently shown success, and have been used in conjunction with neural networks as in [SHJ+15].

At the Interspeech 2016, Shivakumar et al. [**?**] present an approach to Language Identification using a combination of i-vectors, probabilistic linear discriminant analysis, SVM and and Maximum Likelihood Lexical Classification to achieve an acurracy up to 76 % on 12 languages, a result comparable to our results.

2.3. Similar Approaches

Matejka et al. [MZN+14] present an approach very similar to the one introduced in this thesis using Bottleneck Features (BNF) from an ASR trained neural network as input for a LID neural network. Here however a much bigger train data of about 400 hours per language is available in the RATS LID Data Corpus than was available to us, with fewer target languages and a setup where the output of 5 neural networks is averaged, a setup most likely occurring a higher delay than feasible in an online-setting, which is a focus for us.

[LMGDP+14] uses Deep Neural Networks without the previous ASR net setup to identify Language. They compare the approach to the state-of-the art i-vectors as presented above.

[**?**] uses DNNs to recognize different Indian languages that are very similar in nature. They use DNNs with attention in lieu of the recently successful but computational RNNs. They evaluate different net structures and with the best structure find and error rate of only 8%.

Ghahabi et al. in [**?**] use DNNs to recognize the language of short speech samples of below 4 seconds length. They compare their approach to GMM and i-vector systems achieving a relative improvement of around 30 %.

In [HSW12], Heck et al. already evaluate different LID systems for an eventual integration into the Lecture Translator. They evaluate PPRM and PRLM with an SVM classification backend for a Lecture Translator integration. By proposing a hybrid model combining both PPRM and PRLM they are able to further improve upon recognition results by 5%. They however do not evaluate a neural-network based approach as we do in our work.

This work is based on previous work done using LID information to extract another BNF vector to then aid in multilingual ASR [MSW16]. More on this can be found in chapter 4.

3. Fundamentals

The following chapter will define and explain terms and concepts used throughout this thesis, in order to lay a sound theoretical foundation for this work.

3.1. Theory of Language Identification

Language Identification or Recognition, as loosely taken from [LML13], can be formulated in the following way: Assuming we have an audio recording of unknown source language, we convert it into a sequence of acoustic vectors $S = \{s(1), s(2), \ldots, s(T)\}$, where $s(t)$ refers to the frame s extracted from the audio at frame number t. If we have a set of possible equally-probable Languages $\{L_1, L_2, \ldots, L_N\}$, with one of the targets being a combining class for non-target languages as to not limit the approach, Language Identification is the task of finding the Language L^O given audio sequence S , such that:

$$L^O = \arg\max_I p(S|L_I) \tag{3.1}$$

Using phones and phonotactic knowledge sources, a tokenizer or similar approaches before trying to identify the language, we assume that each speech sequence can be segmented into a sequence Υ of phones v which means:

$$L^O = \arg\max_I P(\Upsilon|L_I) \tag{3.2}$$

While in 3.2 we assume to know the exact phoneme sequence, in practice Υ has to be retrieved by selecting the most likely phone sequence from all possible sequence. Using Viterbi Decoding on a set of phone models M we can retrieve the optimal Υ^O:

$$\Upsilon^O = \arg\max_v P(S|v, M) \tag{3.3}$$

Combining 3.2 and 3.3 and considering all possible Υ instead of the best hypothesis Υ^O we get:

$$L^O = \arg\max_I \sum_{\forall\Upsilon} P(S|\Upsilon, M)P(\Upsilon|L_I) \tag{3.4}$$

3.2. Janus Recognition Toolkit (JRTk)

The Janus Recognition Toolkit (JRTk) also known as just “Janus” is a general-purpose speech recognition toolkit developed in joint cooperation by both the Carnegie Mellon

University Interactive Systems Lab and the Karlsruhe Institute of Technology Interactive Systems Lab [**?**]. Part of Janus and the JRTk are a speech-to-speech translation system, which includes Janus-SR, the speech recognition component, the main part of Janus used in this thesis.

Developed to be flexible and extensible, the JRTk can be seen as a programmable shell with Janus functionality being accessible through objects in the tcl/tk scripting language. It features the IBIS decoder, that uses Hidden Markov Models for acoustic modeling in general, although in this thesis we used a neural network as our speech recognizer to generate the input features required by our language ID network.

This thesis makes extensive use of the JRTk's and tcl/tk's scripting capabilities to be able to pre-process speech audio files for further use by our experimental setup. It also uses tcl/tk scripts and its Janus API functionality in the development of our smoothing and evaluation scripts as can be seen in ch. 7.

3.3. Neural Networks

Artificial Neural Networks today are used in many different fields: from image recognition/-face recognition in [LGTB97] to natural language processing in [CW08] and, as relevant to this thesis, to Speech Recognition with very successful examples as in [HDY+12]. It has also been used in the realm of Language Identification, which has been described in the previous chapter. This section will provide fundamental knowledge of how neural networks work and how to train them, to lay the foundation for the explanations of our experimental setup and evaluations. The information in this section is based on [HN04], [GBC16] and [Nie15], if not otherwise sourced.

3.3.1. Setup

Neural networks are based on collections of small "neural units" working together in tandem. The neuron's behavior can be loosely linked to the brain's axons. Each neuron is connected with others and a neuron is "stimulated" by input on these connections. It then decides on its own activation, or stimulation, by using a summation, or threshold, function with a certain limit to decide if the neuron "fires". Finally its own activation is propagated through the network to adjacent units. By changing weigths and activation thresholds in the network, its output changes, therefore the possible adjustable parameter set Θ for a neural network includes all the weights for all neurons as well as all thresholds for the activation functions in each neuron.

3.3.2. Artificial Neuron

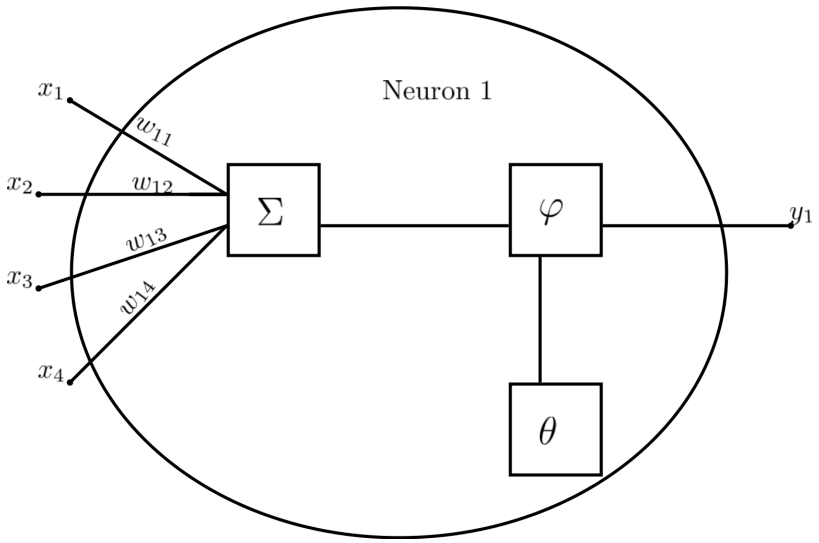
An artificial neuron, or perceptron in its most basic form, is a mathematical function that consists of four parameters that can be adjusted independently from each other:

- w_i the input weights for all inputs
- Σ the transfer function for summation of the weighted inputs
- φ the activation function that calculates the output value y_k based on the transfer input and the threshold
- θ the threshold which defines when the neuron activates.

This means an artificial neuron with output y_k is the function 3.5. Many of these neurons coupled together (via the output of a neuron on a previous layer becoming the input for one on the current layer), make an artificial neural network as used in this thesis. A schematic drawing of this can be seen in fig. 3.3.2.

$$y_k = \varphi(\sum_{j=0}^{m} w_{kj} x_j) - \theta_j \tag{3.5}$$

Figure 3.1.: A schematic drawing of a Neuron and its parameters



3.3.3. A Generic Neural Network

A basic neural network consists of three layers: the input, a hidden layer of neurons and the output layer. Hereby, the output layer consists of as many neurons as classes that the network is trying to classify against and the one with the highest activation after input has propagated, is the classification output of the net. A basic, fully-connected (referring to the connections between neurons, so fully-connected means each neuron is connected to each possible other neuron) net can be seen in fig. 3.2.

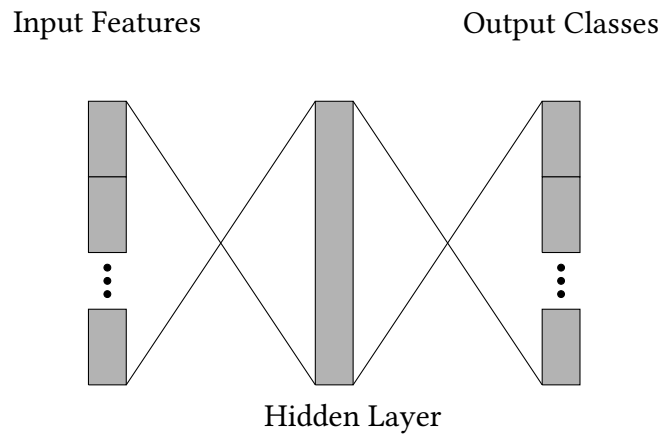


Figure 3.2.: Basic Feed-Forward fully-connected Neural Network.

3.3.4. Network Types

3.3.4.1. Feed-Forward Neural Networks

A basic (non-deep) *Feed-Forward Neural Network* consists of three layers: the input, a hidden layer of neurons and the output layer. Feed-Forward refers to the fact, as opposed to *Recurrent Neural Networks*, connections between the neural units are only in a *forward* direction. This means that layer l_i, preceded by layer l_{i-1} and followed by layer l_{i+1}, will only have connections towards l_{i+1} but not towards l_{i-1}.

Basic non-deep neural networks were the first ones employed in machine learning, especially since computational power at the time was not good enough for the training of multi-layered neural networks and were (erroneously) believed to have limits [**?**]

3.3.4.2. Deep Feed-Forward Neural Networks

Deep Feed-Forward Neural Networks, DNNs, the net-type most used in this thesis, refer to networks that have more than one hidden layer between input and output, but still feature non-cyclic connections between neurons. DNNs have a better performance than single-hidden-layer-networks in general, but require different techniques for training.

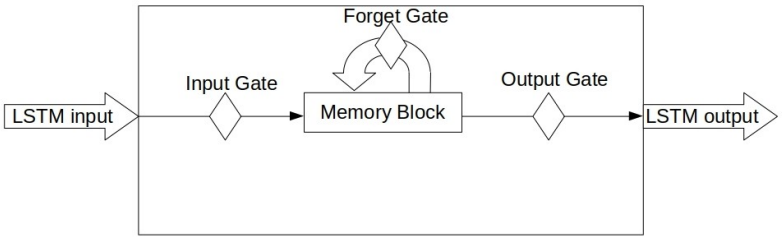
A common description of this phenomenon is, that each hidden layer increases the level of abstraction the network can manage. E.g, in image processing, if the first layer recognizes a color in a certain pixel, then the next layer can infer more abstract characteristics from the output of the first layer. So, after knowing that a certain pixel is dark, the next layer can derive that area might be the eye in a picture of a face, etc. This obviously makes more complicated classification tasks possible but also makes learning algorithms more difficult.

3.3.4.3. Deep Recurrent Neural Networks

Deep Recurrent Neural Networks, RNNs, refer to neural networks that are DNN's but backwards connections (so l_i can have connections both to l_{i-1} and l_{i+1}) are allowed. This means the network can have temporal behavior, as its performance changes dynamically over time, when the state of neurons in later layers changes and affects the neurons in preceding layers.

Long-Short-Term-Memory (LSTM) The most common RNNs are based on LSTM units, artificial neurons that can remember values for a short or long duration. They thus have an internal state that changes over time and are not only decided by a threshold as normal artificial neurons are. Normally, RNNs then include LSTM blocks of multiple LSTM units. The LSTM units then control the change of memory through "gates". The input gate, to control when a new value is entered into memory, the forget gate, to control if a value remains in memory, and the output gate, which controls the calculation of the activation of the entire block. An overview can be seen in fig. 3.3.4.3 The weights that are learned for LSTM blocks are then used in front of the gates. That means the weight determines the values of the memory in the block and its weighting compared to the input and therefore the activation. RNNs are trained with a similar algorithm as introduced in the next section for non-recurrrent networks, called backpropagation through time. A detailed explanation of this can be found in [Wer90].

Figure 3.3.: A schematic drawing of a LSTM Unit with its gates.



3.3.5. Learning

The interesting part about neural networks is their ability to learn from data and improve their own performance. Improving performance in this case means that by adjusting the available parameters of the neurons part of the network, we minimize a cost function that describes the difference between an optimal output and the actual output.

A network can be seen as an approximation of a function f^*. So, a network trying to classify an input x into a class y approximates:

$$y = f^*(x) \tag{3.6}$$

Then one run of the network with parameter set Θ gives the mapping $y = f(x;\Theta)$ and we are trying to minimize our cost function of $C = f^* - f$ by adjusting the set of parameters in Θ each run.

Three basic approaches exist for training a network:

- *Supervised Training*, where the optimal output for train data is known. This means, the train data has been pre-classified by a teacher. This is the method we use in this thesis and it is further explained below.
- *Reinforcement Training*, where the optimal input for train data is not known prior to training, but the environment gives the net feedback about its own output and good output is reinforced while bad output is discouraged.
- *Unsupervised Training*, where nothing is known about the environment and the net (often) just tries to learn the probabilistic distribution of the data.

3.3.5.1. Sampling

Data for training is generally split into three sets: the training set with a size of about 80% of the total available data, which is then used for the training, the development set with a size of about 10% used for evaluation of the net and adjusting different parameters without "distorting" the last test set which also has a size of around 10% and is used for final "clean" validation of the net performance.

Sampling for the three sets is most commonly, as in this thesis, done using *Simple Random Sampling* (SRS), where each set is chosen randomly without bias, so that each sample has the same chance to be part of any of the three sets, and furthermore any combination of n samples has the same probability to be in any of the sets [Men13]

3.3.5.2. Supervised Training

Supervised Training refers to training where the optimal output for the training data is known, so a classification of the train data exists prior to training. This makes calculation of the cost function as defined in the previous section relatively easy, as we define C as the actual difference in output of the current net with current parameter set Θ compared to the teacher-defined classification/labeling.

Mean Squared Error Function One way to calculate the difference between the net and the teacher-classification is by using the mean squared error, as is used in the training of nets in this thesis. If t is the expected output, $f(x;\Theta)$ is the actual predicted output and M the number of output neurons/classes, we define the Mean Squared Error (MSE) as:

$$E = 1/2 \sum_{j=1}^{M} (f(x;\Theta) - t)^2 \tag{3.7}$$

Stochastic Gradient Descent The goal of course is to minimize the MSE as defined above by adjusting parameters for each neuron and layer (including weights to output neurons) to change the predicted output of the net. This is done by adjusting the weights in direction of the falling gradient for each weight.

$$w_{i+1} = w_i - \eta \nabla E(f(x; w), t) = w_i - \eta \nabla 1/2 \sum_{j=1}^{M} (f(x; w) - t)^2 \tag{3.8}$$

This method is called the *Gradient Descent*, with η being the learning rate, the freely adjustable speed at which the network changes and therefore learns. As calculation of this gradient for every sample in the training data set is expensive, a stochastic approach is used where only a small number of samples is used each iteration to calculate the gradient, as the relation between the change of the mean error and the value of samples is not linear. Therefore the calculation of the *Stochastic Gradient Descent* (SGD) is enough to estimate the real required parameter-changes.

Backpropagation While the SGD is used to calculate each iterative weight according to the falling gradient, the Backpropagation algorithm is used to calculate each weight from the total output of the net and its input. If we have neural network output vector (all output neurons together) t_i, input vector (all input dimensions together) x_i and current weights w_i we can calculate w_{i+1} by calculating the SGD on the function $w \rightarrow E(w, x_i), y_i)$.

With preceding definitions we can summarize the training algorithm of a DNN as follows:

```
1  w_0 := rand()
2   do
3      forEach train-sample s
4         f(x;w_i) = neural-net-output(w_i,s) // Actual calculation using each
              neurons output -> Forward pass
5         t = pre-classification(s)
6         Error = E(f(x;w_i),t)
7          w_{i+1} := w_i - η∇E(f(x;w),t) = w_i - η∇1/2 Σ_{j=1}^{M} (f(x;w) - t)^2  //For all Weights and
               layers -> Backwards pass
8         w := w_{i+1}
9    if (ΔE <= Threshold) break
10 return Net
```

Listing 3.1: Pseudo Code to show the Backpropagation/SGD algorithm in Action

4. Experimental Setup

This chapter lays out the experimental setup used in this thesis. While Language Identification is applicable in many different scenarios, here the focus lies on trying to establish a low-latency online approach for recognizing the spoken language in a university-lecture environment. Because finding a suitable test setup for online data retrieval is hard, the data used was cut to short lengths to make an evaluation as to correctness of the recognition possible in an ”online-like” scenario. This means that the output of the net is evaluated after short samples of speech and therefore can be seen as indicative of online performance of the net in the Lecture Translator. The test setup will be introduced in chapter 7

This chapter introduces our experimental setup and tooling, and gives an overview of the used data sets.

4.1. Language Identity Neural Networks

As this thesis continues the work of [MSW16], the experimental setup stayed mostly similar. After extraction of the audio features, as further described in Ch. 5, we get a feature vector of 702, consisting of a combination of IMel and tonal features with a context of 6 frames as input and CD phoneme states as targets. This was trained in the mentioned previous work in a multilingual fashion, meaning it consists of multiple output layers, one for each layer.

The second last layer is the bottleneck layer (referring to a layer with a much smaller size than the previous and following one) with a size of 42. We then use the Bottleneck activations in training for the LID Neural Network. These BNF Vectors are stacked with a context to provide the input for the 2^{nd} LID network, as language identity is considered to be a long-term property of the audio data. We tried different context sizes (see sec. 5.4), choosing the most robust one. In opposition to the previous work mentioned where another BNF layer was introduced in the LID network to reuse the BNF vector in speech recognition tasks, we then use the output layer of the 2^{nd} layer to determine the language of the audio. This means the output layer consists of as many output neurons as available languages.

fig. 4.1 gives an overview of this setup.

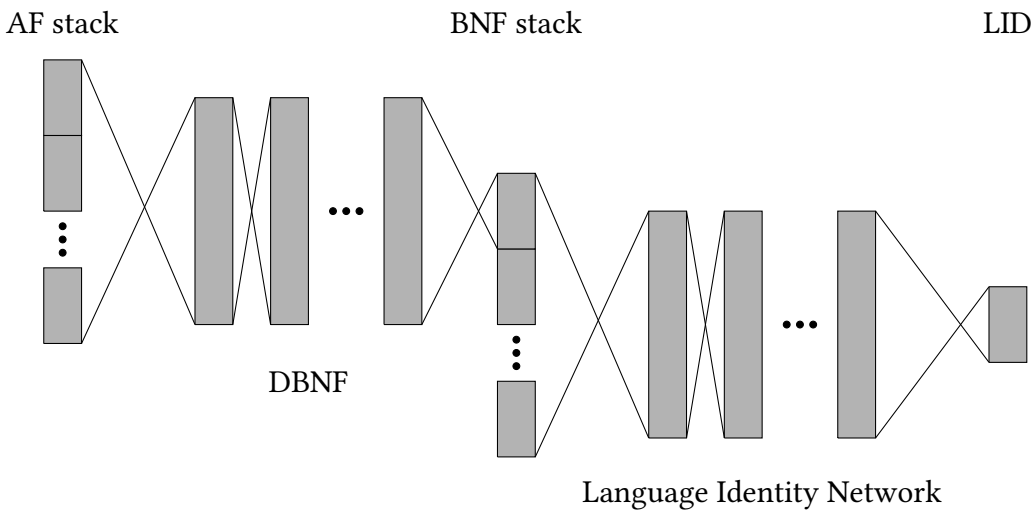


Figure 4.1.: Overview of the network architecture used to extract the Language Identity (LID). The acoustic features (AF) are being pre-processed in a DBNF in order to extract BNFs. These BNFs are being stacked and fed into the second network to extract the LID.

4.2. Tooling

Many different tools exist for deep learning, the most acclaimed being Tensorflow [AAB+16], DL4J/ND4j[1] and Theano [BBB+11]. Pre-existing work on Language Identification, or rather Language Feature Vector Extraction using a LID Network in [MSW16], used a python wrapper around Theano for training, that was developed by Jonas Gehring [Geh12]. This thesis continues the use of this wrapper for the training of our DNNs. The basic layout of the network used and improved upon by this thesis were input vectors from a multilingual ASR net, further explained in sec. 5.1, which we used to generate Bottleneck Features with 966 coefficients. The LID network introduced in the following chapters then uses these BNF vectors to classify the audio input into a language. Different Network Structures and types (DNNs/RNNs) were used that will be further explained in Ch. 6.

As `detl` does not provide support for RNNs we used Lasagne [2], another wrapper around Theano, as the framework for our RNN training.

4.3. Euronews 2014

The first data set used, was retrieved from Euronews [3] 2014. Euronews is a TV channel that is broadcast in 13 different languages simultaneously both on TV and over the Web and is semi-automatically transcribed. The data corpus includes 10 languages (Arabian,

[1] DL4J: `https://deeplearning4j.org/index.html`
[2] Lasagne: `https://lasagne.readthedocs.io/en/latest/`
[3] Euronews: http://www.Euronews.com/

German, Spanish, French, Italian, Polish, Portuguese, Russian, Turkish and English) with around 72 hours of data per language provided overall, meaning the corpus had a total length of around 720 hours of audio data. This data was taken both from online video and recordings of the transmissions as described in [Gre14].

For the purposes of this work we then broke this corpus down further into a more manageable size as to make the feedback-cycle faster, while still keeping the corpus big enough to make results comparable to performance with the full corpus. Later, we then used the full data set to compare results between the smaller and bigger sets.

The corpus was broken down to a per-speaker-basis based on its automatic transcriptions. From this data we took a sample of a random 10.000 speakers, while making sure the total length of samples for each language were roughly the same. Details of this breakdown can be seen in table 4.1.

Table 4.1.: The Euronews corpus speaker breakdown used for most experiments with total utterances length.

Language	**Number of Speakers**	**Combined Length**
Arabian	1055	16.76 h
German	928	18.80 h
Spanish	932	18.78 h
French	1016	18.67 h
Italian	935	19.00 h
Polish	1229	18.30 h
Portuguese	1062	16.19 h
Russian	958	18.66 h
Turkish	957	18.61 h
English	928	18.54 h
Overall	10000	182.31 h

Table 4.2.: The Euronews corpus breakdown into the three data sets.

Set	**Number of Speakers**	**Combined Length**
Train	8000	149.76 h
Development	1000	19.46 h
Test	1000	19.29 h

The speaker list was then split into three smaller datasets: the train set, development set and test set using the common Simple Random Sampling. The sizes were 80% allocated to the train set, and 10% to each the development and test set. Table 4.2 shows the split data for the three sets. The development set will be referred to as DEV and the test set as TEST after this.

The complete Euronews Corpus available was much larger. This was used to confirm intuition that a larger data corpus leads to better results which can be seen in sec. 6.3. The breakdown of the large corpus' data is listed in table 4.3.

Table 4.3.: The full Euronews corpus speaker breakdown with total utterances length.

Language	Number of Speakers	Combined Length
Arabian	4401	51.13 h
German	4436	73.21 h
Spanish	4464	75.71 h
French	4434	68.14 h
Italian	4464	77.22 h
Polish	2625	33.15 h
Portuguese	4430	49.07 h
Russian	4418	72.23 h
Turkish	4385	70.42 h
English	4511	72.76 h
Overall	42568	643.04 h

4.4. Lecture Data

As part of the development of the KIT's Lecture Translator, German lectures at the KIT were recorded and annotated. This is described in [SKM+12]. This thesis then uses parts of this German corpus as well as newer recordings done at the KIT of English lectures with the same setup, English academic talks given at InterACT25[4], as well as French talks done at the DGA's [5] yearly academic conference on speech recognition.

This lecture data was then used in two different ways: Firstly, to evaluate the 10-Language trained Euronews-Net(s) to see how it would fare in a lecture-environment as part of the KIT's Lecture Translator by scaling the output down to the 3 available languages. Secondly to train a second net and further test the findings about the net setup and net evaluation, as found with the Euronews corpus.

The breakdown of speakers and length can be seen in table 4.4. As the main work was done on the Euronews corpus this data corpus is considerably smaller and was mostly just used as a proof-of-concept for a possible integration of a LID-Net into the Lecture Translator.

The development set for the LD corpus consisted of 1.5h of Lecture Data per each of the three languages with one speaker for German/English (one class' recording) and 5 French speakers from DGA talks.

4.5. European Parliament

As another form of evaluation, recordings of the European Parliament speeches, that are freely available online [6] were used. The video recordings come with the simultaneous

[4]InterACT25: `http://www.interact25.org/`
[5]DGA: `http://www.defense.gouv.fr/dga`
[6]EU-Parliament plenary speeches: `http://www.europarl.europa.eu/ep-live/en/plenary/`

Table 4.4.: The Lecture Data corpus speaker breakdown with total utterances length.

Language	Number of Speakers	Combined Length
German	8	16.22 h
French	30	8.25 h
English	27	10.78 h
Overall	65	35.25 h

translations into all the official languages of the EU-countries. This includes seven of the languages also available on Euronews, namely German, English, French, Spanish, Italian, Polish and Portuguese. We extracted the seven audio tracks embedded in the recordings with ffmpeg [7]. The audio extracted is of the simultaneous translations with an underlying audio track of the original speaker, which adds noise, but is similar in nature to the Euronews recordings that often feature an underlying audio track in a different language.

The small corpus was used to evaluate performance of the Euronews-trained nets, as well as improve capabilities of the networks with cross-training on top of the Euronews-nets, of which results can be seen in the following chapter.

As each parliament discussion includes the same audio tracks, the length of all the languages in the set is the same, with each language having 3.6 hours of data.

[7] ffmpeg: `https://ffmpeg.org/`

5. Feature Preprocessing

This chapter deals with the feature preprocessing used to form normal speech into feature vectors to be understood by neural networks. The setup used is based on the standard capabilities of the Janus Recognition Toolkit. The following sections describe this Feature Preprocessing for data as well as the first multilingual ASR network as introduced previously, used to create the BNF features the LID net requires. It is based on preprocessing introduced in [MSW16]. Features produced by this preprocessing setup are then written to "pfiles" together with their labels (the target language). The pfiles are then used to train the network using detl, or, in the case of RNNs, lasagne by supervised training employing the Stochastic Gradient Descent with Newbob scheduling.

5.1. Feature Retrieval

The audio files available to us in the Euronews and Lecture Data corpus were recorded using a sampling rate of 16 kHz. European Parliament data we collected ourselves were recorded with a sample rate of 44.1 kHz. We down-sampled the data using soX[1].

Full Euronews Corpus The Full Euronews Corpus, as introduced in section 4.3, had prexisting pfiles from previous work [MSW16] that we reused. This meant that the features in this case were completely based on 32ms windows and not a combination of 16ms and 32ms windows as for the rest of the corpora.

5.2. Feature Description

For the DBNF network as input, a combination of sound power, lMEL, fundamental frequency variation (FFV) [LHE08], [LHE08] and pitch acoustic features [Sch99] was used. Tonal features have shown improvements for DNNs, even with non-tonal languages as English [MSW+13], so they were also incorporated. Lst. A.1 in the appendix shows the complete feature extraction source code in tcl.

Using the sound power feature that is part of Janus' standard capabilities, we added a speech-detection feature, which identifies speech and noise from the audio. We first calculate the *log* on the current sound power and then determine the weighted average with a context of 2 frames on each side. After normalizing the resulting value between

[1] soX Sound eXchange: `http://sox.sourceforge.net/`

-0.1 and 0.5, we apply a basic threshold function: We consider everything above 0 in the normalized spectrum as speech, and below as (non-speech) noise. We do not discard the non-speech marked parts however, but weigh the combined feature vector less in the resulting input vector for the neural network.

The log mel features are calculated the following way: We calculate the spectrum of the audio waveform using the Fast Fourier Transformation on 16 ms windows, calculate the Mel Coefficients, apply Vocal Tract Length Normalization (VTLN) on the spectrum in the linear domain and then multiply the Mel-Filterbank coefficients with the normalized coefficients. We apply a logarithm to this to get the lMEL features.

From Janus' standard pitch capability, calculated on 16 ms windows, we calculate the symmetric delta coefficients $(f(t+\delta) - f(t-\delta)$ on the pitch for $\delta = 1, 2, 3$ and the same deltas on those results and merge to give us a coefficients for frame t. This means we define:

$$f(t) := PITCH(t+1) - PITCH(t-1) \tag{5.1}$$

$$g(t) := PITCH(t+2) - PITCH(t-2) \tag{5.2}$$

$$h(t) := PITCH(t+3) - PITCH(t-3) \tag{5.3}$$

Merging these we get the coefficients:

$$(PITCH(t), f(t), f(t+1) - f(t-1), g(t), g(t+2) - g(t-2), h(t), h(t+3) - h(t-3)) \tag{5.4}$$

The FFV is calculated on 32 ms windows, to then be merged with the pitch feature we just defined to make up the TONE feature. lMEL coefficients and TONE coefficients are then merged. See [**?**], [LHE08]These feature vectors are stacked with a context of 6 adjacent frames and then passed to the DBNF net as introduced in the next section.

Full Euronews Corpus The full Euronews corpus used 32 ms for all feature calculations, so the sound power, FFT/lMEL and pitch were calculated on 32ms windows, but used the same features aside from this.

Lecture Data/European Parliament For the other two data corpora we kept the 16ms windows for power and lMEL, but changed the pitch to be calculated on 32 ms windows.

5.3. DBNF network

The DBNF was trained as part of [MSW16]. It is trained using the input features defined in section 5.2. The targets are context-dependent phoneme states, representing a similar setup to how an ASR neural network. The network has 6 layers of 1000 neurons each plus a bottleneck layer of 42 neurons before the final output layer.

The activations of the bottleneck layer are extracted. We then apply a context of 11 frames and stack the frame itself with the context. The context is calculated using a spread of 3. This means we only use every 3^{rd} frame. The stacked vector is then used as the input for the LID network on which the following chapter will elaborate.

5.4. Evaluating Input Features

Work in [MSW16] included a comparison of different context spreads as stacked inputs for the LID net of 2,3 and 6. As one of the first experiments on the Euronews corpus we increased the spreads from 3 to 4 and 5 frames, while still keeping the original context width of 11 (meaning a total context width of 44 Frames (880ms) and 55 Frames (1100ms) respectively). This however proved not successful as performance decreased already in the frame-based metrics and was therefore likely to also decrease in the per-sample metric as introduced in the following chapter.

For all further experiments on Network layout and smoothing, we therefore used the best performing context of 33, with a spread of 3 frames.

table 6.1 at the end of the next chapter shows the results for the different selected spreads with different net structures, which are introduced in the following chapter.

6. LID Network Structure

This chapter describes the actual Language Identification neural networks trained as well as the results of different network/data setups used. Most network experiments in the network geometry, referring to the number of hidden layers as well as the layout of the neurons in these layers, were evaluated using the Euronews corpus. Results from this corpus were then transferred over to the other corpora, meaning that the network layout that worked best for Euronews was then adjusted for the lecture data but otherwise the geometry was kept intact.

6.1. Baseline Setup

The first experimental net setup consisted of 5 layers of denoising auto-encoders. Each layer had 1000 nodes, aside from the input layer using the 966 feature frames that is the input vector whose contents were described in the previous chapter, and a `tanh` activation function using the mean squared error as the loss function.

The neural net was then trained using mini batches of size 2,000,000. The training was done using a learning rate of 0.01. The pre-trained net was then retrained with a 1000 neuron to 10 coefficient output to get to our 10 Languages as classes to classify against. In the basic setup this was trained using a learning rate of 1 and the exit condition of a minimum change of 0.005 / 0.0001 for the training/validation data respectively.

The beginning benchmark to improve upon was then set to the frame-based validation/-train error of 0.23 / 0.27 respectively, while of course understanding that non-training per-sample data would most likely have worse results at first than the frame-based validation error calculated on training data.

The following section describes different network layouts and changes we made to the training of the neural network and the improvements made upon our initial result.

6.2. Improving Network Layout

Different network layout were tried and experimented with. This includes a 6 pre-trained hidden layer version, as well as changes in the geometry. The differences in the frame-based errors can be seen in table 6.2.

Decreasing Learning Rate Setting the learning rate of the output layer training to 0.1 instead of the default 1, gave us the first increase in frame-based recognition rates, improving it from an error of 0.274 for the base setup to an error rate of only 0.256, giving an improvement of almost 2%. As table 6.2 shows, the higher learning rate lead to a better train-set result but worse validation performance, this could be an indication that the lower learning rate stops the net from over-learning as much and increases generality.

Adding the 6th hidden layer Adding another 1000 neuron hidden layer did not lead to an increase in recognition compared to the version using a lower learning rate and rather gave us an decrease of about 2%, meaning an error rate of 0.275 compared to the 0.256 for the lower learning rate version.

"Tree" net structure The biggest increase was achieved by changing the geometry of the net from 5 layers with each 1000 neurons to one of each layer having 200 less neurons of the previous one giving us a net definition of (`tanh` being the activation function, `mse` the loss function of the mean squared error and make da the command for `detl` to add another hidden layer):

```
'!make da 966,tanh:1000 loss=mse' \
  '!make da 1000:800' \
  '!make da 800:600' \
  '!make da 600:400' \
  '!make da 400:200' \
```

This setup, with the standard output layer with 10 output neurons, improved error rates from 0.256 down to 0.245, an improvement of 1.2 %.

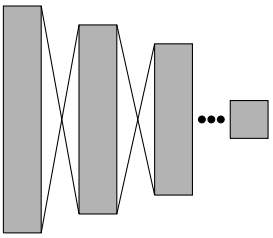
Adding another 6th hidden layer onto this structure to give a net definition of (visualized in fig. 6.1):

```
'!make da 966,tanh:1200 loss=mse' \
  '!make da 1200:1000' \
  '!make da 1000:800' \
  '!make da 800:600' \
  '!make da 600:400' \
  '!make da 400:200' \
```

further improved the results, yielding an error rate of only 0.241, performing another 0.4 % better than the 5-layer tree-structure.

A possible explanation for the better performance of the tree structure could be, that the farther the layer is removed from the input, it works on recognizing more "abstract" patterns, and less of the more abstract classes exist than of basic patterns. E.g. the first layer decides on the values of one certain frequency of the input what language is more

Figure 6.1.: Visualization of the tree-net structure.



likely, with the second layer then taking into account a whole range of frequencies (of which less exist) or even more abstract features to further differentiate.

Adding a larger number of hidden layers into this tree structure did not improve results, as a 10-hidden-layer version with the setup of:

```
'!make da 966,tanh:2000 loss=mse' \
 '!make da 2000:1800' \
 '!make da 1800:1600' \
 '!make da 1600:1400' \
 '!make da 1400:1200' \
 '!make da 1200:1000' \
 '!make da 1000:800' \
 '!make da 800:600' \
 '!make da 600:400' \
 '!make da 400:200' \
```

failed to perform featuring an error rate of 90 %, meaning the net was not able to classify correctly anymore.

As deeper hidden layers will take longer/more data to train correctly (as the changes have to propagate through the earlier layers first, to then affect the deeper layers), our setup did most likely not include enough data for the training of these nets. Another explanation could be the spreading of information in the input 966 features onto the 2000 neurons of the first hidden layer added wrong coefficients.

Overall the tree-net with 5 layers and reduced learning rate performed the best in the frame-based errors, featuring a relative improvement of 0.175 compared to the basic net.

6.3. Full Euronews Corpus

As introduced in sec. 4.3, a bigger Euronews data was used to further evaluate our network structure. The net was trained with the tree-net 6-layer structure with a reduced learning rate as introduced above. This resulted in a train error of 0.204 and a validation error of only 0.169, an improvement of 6.9% compared to the smaller corpus with the same net structure. However, it only improved the error in classifying each sample of the DEV set

Table 6.1.: Results of the different context sizes on the Euronews corpus for the first 3 introduced net structures. Frame-based errors on train/validation set.

Net Structure (context)	**Train Error**	**Validation Error**
Basic (33)	0.226	0.285
Basic (44)	0.073	0.353
Basic (55)	0.209	0.309
6-Layers (33)	0.236	0.276
6-Layers (44)	**0.192**	0.326
6-Layers (55)	0.243	0.298
Reduced Learning Rate (33)	0.273	**0.266**
Reduced Learning Rate (44)	0.287	0.310
Reduced Learning Rate (55)	0.276	0.268
Best	**0.192**	**0.266**

Table 6.2.: Results of the different net structures on the Euronews corpus. Frame-based errors on train/validation set.

Net Structure	**Train Error**	**Val. Error**	**Rel. Impr. Val. Error**
Basic (5 layers 1000 Neurons)	0.226	0.285	-
6-Layers (1000 Neurons)	0.236	0.276	0.032
Reduced Learning Rate	0.273	0.266	0.067
Tree Structure (5 layers 1000...200 neurons)	0.221	0.245	0.140
Tree Structure (5 layers 1000...200 neurons) and reduced learning rate	0.245	**0.235**	**0.175**
Tree Structure (6 layers 1200...200 neurons)	**0.211**	0.242	0.151
Tree Structure (6 layers 1200...200 neurons) and reduced learning rate	0.251	0.238	0.164
Tree Structure (7 layers 1400...200 neurons)	0.206	0.247	0.133
Tree Structure (10 layers 2000...200 neurons)	0.899	0.910	-
Best	**0.211**	**0.235**	**0.175**

(as further introduced in the next chapter) by about 4%. As the amount of train data was trippled while the classification only improved by about 4 %, this suggests that a suitable post-net filtering approach and net setup will have a bigger impact than the data corpus size after a certain threshold.

Table 6.3.: Comparison of the big and small Euronews data corpus. Frame-based errors on train/validation set, as well as Sample-based Error.

Data Corpus	Train Error	Validation Error	Sample Error
Small (17h/Language)	0.251	0.238	0.318
Full (70h/Language)	0.204	0.169	0.276
Change	**0.047**	**0.069**	**0.042**

6.4. Cross-set training

As a further experiment, we used the pre-trained nets of the stacked hidden layers of the Euronews net and trained the output layer on top with the cross-set of Lecture-Data. This was done by loading net-parameters from pre-training and training the final output layer with the cross-set on top of them. For the final layer we used a low learning rate of 0.1 (1/10th of the learning rate we used in the normal setup), to make meaningful learning possible while cross-training only the final layer.

As a basis for the cross-training, we loaded the tree-net 6-layer structure of the Euronews nets, omitting the output layer. We then used the lecture data corpus to train the output layer with only 3 output neurons. This yielded an improvement of 4.9% for the error of the tree-net Euronews net recognizing the lecture data development set on a per-sample basis (for which the test setup is introduced in the next chapter).

A second experiment was conducted, using the same base Euronews net, but adding 2 layers on top for the cross-training with 200 and 100 neurons respectively, instead of only one output layer with 200:10 neurons. This further improved the Frame-based results, by another 1.4%., and sample-based results by another 1.8%. This gives us a total improvement of 6.7% for the cross-training compared to the base Euronews net. See tab. 6.4 for the detailed results.

Since the Lecture Data corpus only has 3 languages, we could not evaluate the cross-trained nets using all Euronews languages but still gave a figure for the net results with the Euronews corpus for English, German and French. These results are obviously much worse than pure-Euronews-trained nets, as the two output layers have never seen euronews data, that while similiar, is obviously still sufficiently different, as the performance suffered by 12.1%.

6.5. Combining Lecture Data and Euronews

By combining both the lecture data and Euronews corpusses, we produced improved results that can be seen in Tab 6.5. This was to be expected just from the amount of train data almost doubling for the 3 lecture data-languages. Overall, this net produced the best results for the lecture data corpus by a big margin of 23 %. The concateneted corpus also produced the best results on the Euronews corpus with an increase in the error of only 0.005 compared to the non-augmented Euronews data.

Table 6.4.: Results of the cross-training attempts of the Euronews-trained 6-layer tree-net with the lecture data (LD) corpus. 3 Languages (3L) are German, English, French.

Net Structure	**Euronews (DEV) 3L Error**	**LD Train Error**	**LD Val. Error**	**LD Sample Error**
Tree-net Euronews net w/o cross-training	0.291	-	-	0.179
Tree-net Euronews net with cross-training	0.456	0.075	0.116	0.130
Tree-net Euronews net with cross-training 2 layers	**0.413**	0.073	0.102	**0.112**
Change (best)	**0.121**	-	-	**0.067**

6.6. Lecture-Data-Training

After trying cross-training and concatenation for the lecture data corpus, we also trained a net on the Lecture Data corpus. We continued the use of the tree-net structure as it had proven successful with the Euronews corpus. We first evaluated the same 6-layer setup with and without an adjusted learning rate. Additionally we also experimented with a smaller, 4-layer tree-net with a structure of:

```
'!make da 966,tanh:800 loss=mse' \
'!make da 800:600' \
'!make da 600:400' \
'!make da 400:200' \
```

The 6-layer net did prove to perform better than the 4-layer one in the frame-based metric, and surprisingly in this case, the non-adjusted learning rate version (so the standard learning rate of 1.0) performed better than the lowered one, as the validation error of the 6-layer-standard-lr-net was about 0.6% better than the version with the lowered learning rate for the LD Evaluation. Overall however, the LD-Trained Net with 4 layers performed the best on full samples, the metric that can be most related to performance in an online fashion. This confirms our findings on the Euronews corpus of a tree-net structure performing the best for LID.

Suprisingly, the other ld-only nets did feature a better validation/train error frame-based but failed to achieve higher accuracy in sample-evaluation in comparison with the cross-trained versions. Possibly, this can be attributed to the fact that the lecture data corpus overall is much smaller than the Euronews one, on which the cross-trained version were pre-trained on, and therefore have less transferabilty.

table 6.5 shows a comparison of the different Lecture Data experiments and a summary of the improvements we made. Overall the best net for lecture data would be the concatenated data corpus trained net with a sample error of only 0.65%. However, zhe LD-only trained

Table 6.5.: Results of the different lecture data (LD) approaches. 3 Languages (3L) are German, English, French.

Net Structure	Euronews (DEV) 3L Error	LD Train Error	LD Val. Error	LD Sample Error
Tree-net Euronews net w/o cross-training	0.291	-	-	0.179
Concatenated Euronews/LD-trained net	**0.296**	0.206	0.245	**0.065**
Tree-net Euronews net with cross-training	0.456	0.075	0.116	0.130
Tree-net Euronews net with cross-training 2 layers	0.413	0.073	0.102	0.112
LD-Trained Net 4 layers	0.438	0.076	0.109	0.088
LD-Trained-Net 6 layers	0.439	0.026	0.093	0.150
LD-Trained Net 6 layers & lowered learning rate	0.417	0.066	0.100	0.112
Change (best)	**0.005**	-	-	**0.114**

net with 4 layers and a tree-structure, is only 2.3% worse than this net, while having a much smaller corpus size.

Intuition dictates, that the cross-trained versions would perform better on the Euronews corpus than the ld-only trained ones, which was confirmed by the results. However, the difference in error was smaller than expected, with only 2.6% between the best-performing LD-net and the cross-trained 2-layer net for the Euronews error.

6.7. European Parliament

As to further test the conclusions made about net-structure/cross-training we also trained data on the small European Parliament corpus we collected. The setup was very similar to the Lecture Data tests as introduced in the previous section. We however did not try to concatenate the train data with the other corpora available.

The two cross-training attempts had the same setup. For the EP-only training we found that the 6-layer tree-net structure performs worse than the base setup (5 layers with 1000 neurons each). This however could be due to the minimal data available in the EP corpus and the fact that less layers in this case would always perform better.

Surprisingly, even though the train data available was very small, the improvements in comparison to the Euronews net results for European Parliament speeches are still better by 2.6%, showing that neural networks already start performing even with minimal data available and selection of train data similar to later actual input is very important.

Table 6.6.: Results of the different European parliament (EP) approaches. 7 languages (7L) are German, French, Italian, English, Polish, Portuguese, Spanish

Net Structure	**Euronews (DEV) 7L Error**	**EP Train Error**	**EP Val. Error**	**EP Sample Error**
Tree-net Euronews net w/o cross-training	0.297	-	-	0.383
Tree-net Euronews net with cross-training	**0.676**	0.230	0.292	0.456
Tree-net Euronews net with cross-training 2 layers	**0.676**	0.267	0.310	0.391
EP-Trained Net 5 layers	0.679	0.155	0.333	**0.357**
EP-Trained Net 6 layers	0.691	0.061	0.361	0.392
Change (best)	**0.379**	-	-	**0.026**

See table 6.7 for the summarized results. Overall it's clear, that the European parliament data is substantially different from the Euronews set up as the Euronews trained net performs worse even with cross-training than a net trained directly on the small Europearn Parliament corpus itself.

7. Smoothing and Evaluation

While frame-based error rates on the training/validation sets were already sufficiently good from the LID networks, this of course is not a reliable indicator of real-world online performance, so the development setup consisted of (for each data corpus) a development set (as introduced in Ch. 4) of speakers whose samples were run through the LID setup (Feature Preprocessing ch. 5, DBNF extraction ch. 5.3. LID network ch. 6). The following smoothing approaches were applied in an ”online” fashion, meaning it was made sure they can be calculated in real time while new data is still coming in. The summarized results are listed in tab. 7.10, 7.9 for both the DEV and TEST results. We chose to only run the experiments on the Euronews corpus, as the 10-language-target will always have more possible erroneous outputs and filtering can prove to more successfull. We evaluated different net structures, however the results presented here are based on the tree-net 6-layer structure as introduced in the previous chapter. This is not necessarily the best structure for the frame-based train error, but it gave us the best results on a sample basis (see tab. 7.1).

7.1. Output Activations

We started by evaluating a few samples manually by looking at the output. In the following diagrams one row is equal to activation of one output neuron (bottom to top ID 0-9). Activations are marked black for high activations and white for low activations. The x-scale is the current frame number (with one frame being 10 ms long). As expected activations for random noise/music as in fig. 7.1 are smooth over all the frames and do not show any language as being predominant.

Also, for a 800 ms Polish sample, fig. 7.2 shows the output activations for a correctly recognized sample. It is clear, that the language with ID 5 is the one being spoken in the sample as the 5th row in the picture has the highest activations. This is equal to Polish (as arbitrarily defined in the pfiles writing script).

An incorrectly classified sample can be seen in fig. 7.3. The language of the sample is English (ID 9), but from frame 50 on it is being (mostly) classified as Arabian (ID 0). However, one can also see, that the activations of the network are not smooth and tend to jump around as different languages appear as the maximum in this sample, e.g. also German (ID 1) appears to have high activations at different times.

Figure 7.1.: Activation of output neurons for random non-speech (jingles/noise)

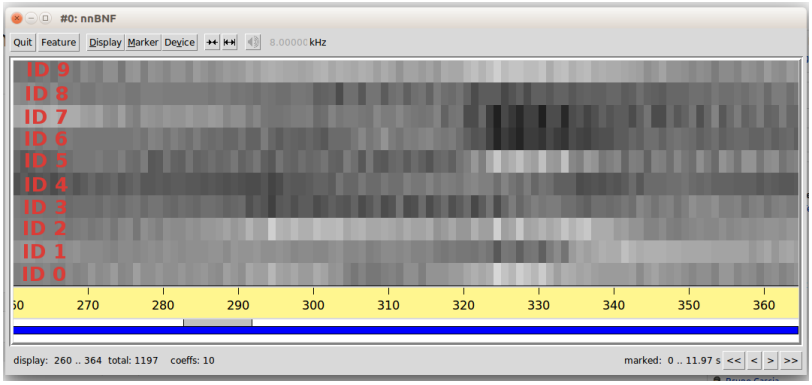


Figure 7.2.: Activation of output neurons for a correctly recognized Polish sample

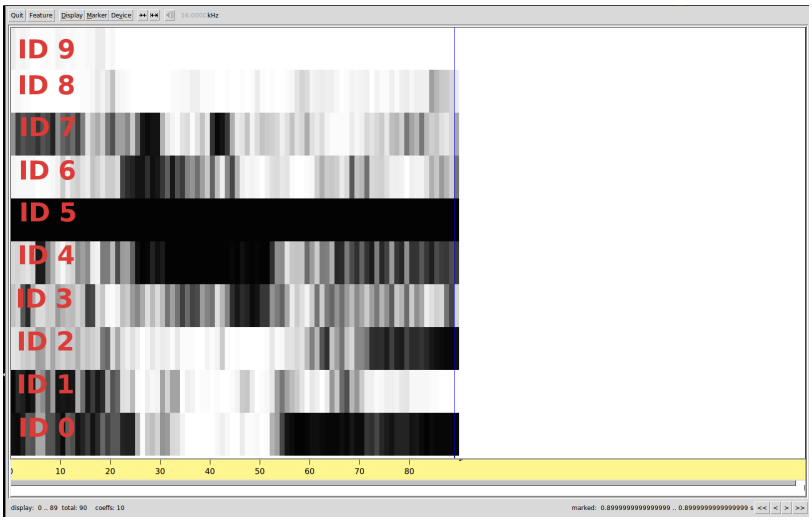
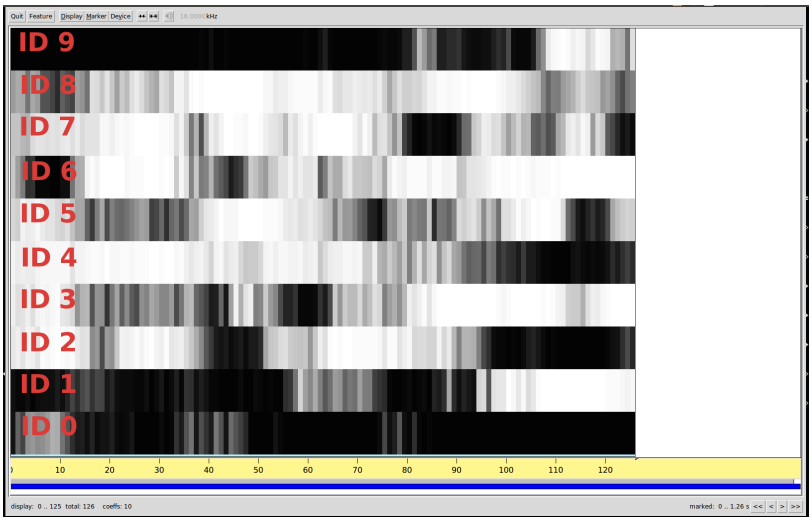


Figure 7.3.: Activation of output neurons for an incorrectly recognized English sample



7.2. Evaluation Metric

To have a baseline to improve upon we evaluated the base net on full samples of differing lengths and then tried to improve these results. See app. A.2 for the corresponding tcl/tk code for this bare approach. It counts a sample as correctly classified if the majority of frames/filtering steps have been classified correctly. The test setup used then goes through the entire set of samples and counts the correctly/wrongly classified samples for each language. This means that an extra amount of smoothing is included, but results should still be sufficiently general to be able to infer properties of employed filters, as they all include this extra smoothing. This additional smoothing can be applied in an online fashion by setting a sufficiently large, while still small enough to not incur delay, window size and then counting the occurence of outputs to give the same effect as used here.

7.2.1. Out-Of-Language Error (OLE)

The first correctness metric proved to be too restrictive for filters. As filters might reduce the count of the correct language while making the total number of wrong outputs less, we present another metric to evaluate filter effectiveness. We count the total number of falsely (as in not-equal to the maximum output) classified frames/output per sample divided by the number of total frames/outputs and average them on a per-sample basis to see if filters actually work in filtering out false outputs. It basically gives a metric for the amount of noise the net output includes. Arguably this would be even more important in an online-fashion in the Lecture Translator, as wrong outputs/filters would be passed on to the speech recognition/translation APIs. This second metric evaluation for well-performing filters can be found at the end of the chapter in table 7.10.

Table 7.1.: Results of different net structures evaluated on the Euronews DEV set.

Net Structure	**Overall Error**
6-layer tree-net lr 0.1	0.318
6-layer tree-net	0.311
7-layer tree-net	0.316

The results of the bare-net evaluation for Euronews can be seen in table 7.2 for the best performing net structure, the 6-layered-tree-structure as introduced in sec. 6.2. The results are, as expected, worse than the frame-based training-recognition, as here the entire (not-yet seen) samples are run through the net. fig. 7.2.1 shows the relation of the length of samples and the amount of correctly classified samples of this length for the tree-structure-net. It shows the intuitive results of a longer length of a sample coinciding with a higher recognition rate through the net, as more frames can be used for the recognition task in the context produced by the ASR BNF net. However, it is also evident that for samples with a length of only 17 frames (200 ms), the net already categorizes the sample better than guessing a language (which on average would yield a correctness of 0.1), as for 17, the net produces a correctness of 0.1896.

As one can see a length of around 100 frames (meaning a sample length of: $100*10/1000 = 1s$) is sufficient to be able to recognize the language with a correctness of ≈70 %. This means online recognition of language is possible with a relatively small warm-up time of 1 second (or around 2 seconds with a higher correctness of then well over 80 %).

Correctness / Sample length for 6-layer tree-structure Euronews net

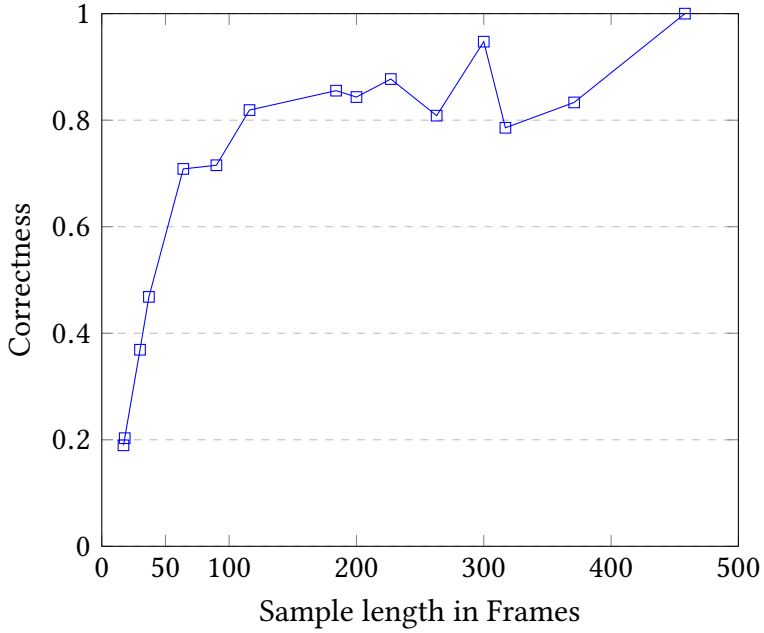


Table 7.2.: Results of the best net structure (tree-structure 6-layer) evaluated on the Euronews DEV with only bare net output.

Language	**Error (total)**	**Error (samples >500 ms)**
Arabian	0.364	0.235
German	0.300	0.155
Spanish	0.320	0.155
French	0.300	0.138
Italian	0.326	0.174
Polish	0.267	0.111
Portuguese	0.291	0.161
Russian	0.315	0.148
Turkish	0.337	0.182
English	0.267	0.145
Overall	0.267	0.162

7.3. Basic Test Filter

The first filter used was basic 5-Frame smoothing: It saves the value of the last direct outputs and only outputs a language if the last 5 direct outputs would have been the same. It also includes a filtering based on the actual output of the language ID neuron, only counting outputs higher than an arbitrarily defined activation of 0.61. However this can be disregarded, as for the vast majority of samples the maximum neuron for a certain frame never fell underneath this threshold.

This approach requires a 5 frame ($\approx 50ms$) "warm-up" time, which still would make it usable in an online environment.

table 7.3 shows the result of the basic filter for each language. Overall the error (for our defined metric) went up by about 5 %. It did however improve results for Arabian by about 3 %, showing that the finding of a suitable filter that improves the overall recognition is non-trivial, as apparently some languages will benefit from a certain type of filter while another will not.

Table 7.3.: Results of the best net structure (tree-structure 6-layer) evaluated on the Euronews DEV Set with the Basic Filter.

Language	**Error (total)**
Arabian	0.337
German	0.312
Spanish	0.331
French	0.313
Italian	0.334
Polish	0.277
Portuguese	0.310
Russian	0.322
Turkish	0.343
English	0.286
Overall	0.318

7.4. Advanced Test Filter

The advanced test filter is based on the JRTk's *FILTER* capability. It automatically takes a defined amount of frames and calculates the weighted arithmetic mean with predefined weights for incoming audio. First tries were done using a small filter setup of:

```
filter          nnFILTERSMALL   nnBNF  {-2 {1 2 3 2 1}}
```

Herein the context is 2 frames on each side of the current frame (the first parameter) with weights 1, 2 , 3 , 2 , 1 for the 5 frames respectively. It however offered no improvement of results, rather a decline in total correctness so different filter approaches were tried:

```
filter          nnFILTERBIG nnBNF {-5 {1 2 3 4 5 6 5 4 3 2 1}}
```

which however did not provide a further increase in correctness. In our tcl filtering, we only then changed the feature to read the frames from, while still then taking the maximum of the filter-feature as the output. Comparison of the two filter can be seen in table 7.4 for evaluation on the DEV set. As compared to the bare net output, the bigger advanced filter offered a tiny improvement for some languages, (0.1% for both Portuguese and English) in correctness but lost out in all other languages. Detailed per-Language results for this and the following filters can be found in app. A.1.

Table 7.4.: Results of the best net structure (tree-structure 6-layer) evaluated on the Euronews DEV Set with the Advanced FILTER

Filter	**Overall Error**
Small FILTER	0.313
Large FILTER	0.316

7.5. Difference Test Filter

As a next approach, the difference between the two most likely outputs was taken into account. We filtered the direct output by only changing from the previous output, if the difference (which we implied to be how sure the net was when determining the language) was bigger than 0.01.

Aside from the general errors with the correctness metric as already outlined in the beginning of this chapter, we also thought one possible reason for a worse performance of this filter could be, that the output of the 2^{nd} most likely neuron is not going to differ from the maximum output on many language pairs: E.g. French/Italian, Russian/Polish, Italian/Portuguese, even if the net predicts one over the other as the difference between the languages is also relatively "small". E.g as presented in [CM05], the difference between English and French is small as the "linguistic score" is relatively high with 2.5.

The full tcl code of this filter can be found in the appendix . The evaluation results can be seen in tab. 7.10

7.6. Counting Filter

This filter included a counting of the occurrence of a certain net output in a frame-range, and then outputting the ID of the language that was recognized the most in that frame-range (while still then employing the same per-sample smoothing as above). The first try was done using a frame-range of 10, later increased further. We listed the results of this filter with different ranges on the DEV set in 7.5. As a further experiment we also used overlapping windows (meaning we count the occurrence in 10 frames and the next 10

frames are not 10 frames further but starts at the 8th frame of the previous window -> overlap of 3 frames).

The full code of this filter can be found in the Appendix.

Table 7.5.: Results of the best net structure (tree-structure 6-layer) evaluated on the Euronews DEV Set with the Counting Window Filter

Filter	**Overall Error**
Counting Filter FR 10	0.313
Counting Filter FR 10, overlap 3	0.323
Counting Filter FR 50	0.316
Counting Filter FR 100	0.313
Counting Filter FR 150	0.314

7.7. Sequence Filter

This filter included a counting of the longest sequence of a certain net output in a frame-range, and then outputting the ID of the language that was recognized for the ongest sequence in that frame-range (while still then employing the same per-sample smoothing as above). This decreased the OLE substantially as can be see in table 7.10, however the filter did increase the overall error by 8%, more than any of the other filters,

The full code of this filter can be found in the Appendix.

7.8. Two-Language Setup

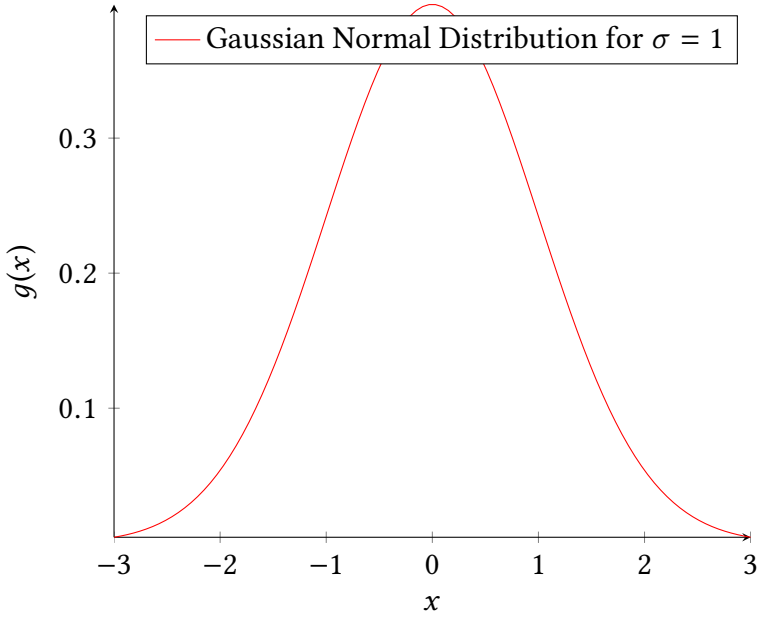
table 7.6 shows the result produced by using the LID Euronews net for 10 languages on different combinations of 2 languages (namely French/Italian and English/German). In this case we ignore the output of the net if it doesn't equal one of the two languages and instead keep the previous output intact in this case. This, intuitively, gave a big boost in the recognition rate, bringing the rate up to 85 % for the two languages combined, an improvement of more than 10 % in correctness compared to the bare approach in sec. 7.2.

7.9. Gaussian Smoothing Filter

Gaussian filters have shown great success in the realm of image processing in the form of canny edge detectors. A one-dimensional Gaussian filter returns a response that is akin to the form of a normal distribution: To calculate the Gaussian Filter Response, one needs a Gaussian Kernel with a Window-Size. While in general the Gaussian filter has an infinite window size, choosing a fixed window size can approximate the Gaussian well. In this

Table 7.6.: Results of the best net structure (tree-structure 6-layer) evaluated on only 2 Languages with the 2-language filter on the DEV set.

Language	**Error (total)**
French/Italian as input only	
French	0.186
Italian	0.113
Overall	0.153
German/English as input only	
German	0.183
English	0.126
Overall	0.156



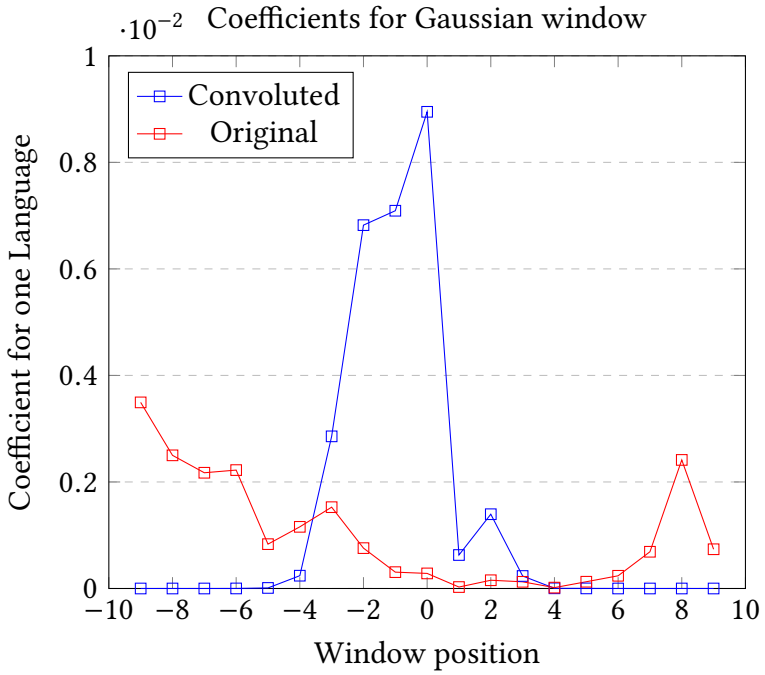
discrete case σ, the standard deviation of the Gaussian, can be approximated with N the size of the window:

$$\sigma = \sqrt{\frac{N}{2\pi}} \tag{7.1}$$

With σ, the Gaussian function for point x can be calculated as:

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} * e^{-\frac{x^2}{2\sigma^2}} \tag{7.2}$$

Once the Gaussian function is calculated for the window, the convolution of the data points with the Gaussian kernel is performed, resulting in smoothed over coefficients of the LID net output. The maximum smoothed coefficient is then chosen and the corresponding language is output, which gave us definite improvements over the unsmoothened net output. In tcl the relevant code part can be seen in A.10, in which we presume the window size to mean the number of points we take into account on both sides of the origin, while normally this number is the window size - 1.



We tried different window sizes of which the result can be seen in tab. 7.7 on the DEV Set. As can be seen, the Window size of 15 performed the best out of the different window sizes evaluated. To showcase a correct implementation fig. 7.9 shows the result of one Gaussian convolution for a window size of 10 as a graph, as one can see it is evident that the graph looks similar to the Gaussian kernel as graphed above and smoothing of the data has evidently occurred. As it turns out the Gauss Filter indeed performs much better even on the correctness-metric on small samples with a size under 50 frames (=500ms). tab. 7.8 showcases this in comparison to the bare net output, which shows a very significant improvement of 15%.

Table 7.7.: Results of the 6-layer tree-structure evaluated with different Gauss filters on the DEV set.

Filter	**Overall Error**
Gaussian Filter WS 10	0.3128
Gaussian Filter WS 15	0.3125
Gaussian Filter WS 17	0.3126
Gaussian Filter WS 20	0.3130

Table 7.8.: Comparison of results of the 6-layer tree-structure evaluated on DEV set for small samples.

Filter	**Overall Error (< 500 ms)**
Bare Net Output	0.590
Gaussian Filter WS 15	0.439

7.10. Speech Filter

As described in Chapter 5 we extract a SPEECH feature from the input audio using the Sound power as a basis. In a further attempt to smooth output we used this to ignore any non-speech marked frames for output-calculation. This only gave us an error of around 0.02% higher compared to the bare net output, however the filtering effectiveness (OLE) was worse than for the other filters as can be seen in 7.10.

Table 7.9.: Results of the 6-layer tree-structure evaluated on samples longer than 50 frames (500 ms)

Filter	**Error (>500 ms) on DEV**
Bare Net	0.162
Counting Filter FR 10	0.178
Speech/Noise Filter	0.163

Table 7.10.: Results of the 6-layer tree-structure evaluated with all the tried post-filtering approaches on both the DEV and TEST set.

Filter	**Overall Error DEV/TEST**	**OLE DEV/TEST**
Bare Net	0.311/0.305	0.202/0.198
Advanced Filter (small)	0.318/0.312	0.185/0.181
Difference	0.317/0.311	0.170/0.165
Gaussian Filter (WS 15)	**0.313**/0.307	0.181/0.176
Counting Filter (WS 100)	**0.313**/0.307	**0.007/0.006**
Speech/Noise Filter	**0.313/0.306**	0.187/0.181
English/German	0.116/0.216	0.145/0.166
French/Italian	0.207/0.214	0.158/0.159
Sequence Filter (WS 10)	0.393/0.382	0.031/0.029
Best (w/o 2L, Bare)	**0.313/0.306**	**0.007/0.006**

7.11. Filter Selection

Overall we conclude, that selection of a good filter is a non-trivial problem, as already we saw that some filters perform better for certain languages or setups while suffering for others. Overall, our net performs reasonably well with all filters, reaching error of less than 20 % for samples of length > 500 ms (See tab. 7.9), a delay that would still make online-usage feasibly. On 10 target languages, using the Counting Filter with a Window size of 100 (so a delay of 1 second), would arguably perform the best while still filtering out the maximum of wrong outputs, featuring an OLE of only 0.007, a relative improvement of $(0.202 - 0.007)/0.185 = 0.97 = 97\%$ for the DEV set and $0.198 - 0.006)/0.198 = 0.97 = 97\%$ compared to the bare net output.

For less delay, one could also implement the Sequence Filter for a small Window size of 10, which also gives us an relative improvement of $(0.202 - 0.031)/0.202 = 0.85 = 85\%$ on the DEV and $(0.198 - 0.031)/0.198 = 0.84 = 84\%$ on the TEST set. However, this filter is likely to possibly change the overall output. However a combination of the Sequence and the Counting Filter on smaller window sizes might be feasible.

The Gauss Filter might also be a feasible choice as it improved results for very short samples (< 500 ms) and less jumps in the output of the Lecture Translator would be preferable, however the computational complexity of this filter is much higher compared to the other filters which might have to be taken into account in an online-environment.

Overall, in the lecture-translator environment, less target languages are likely, and so the language-filter for the correct amount of target languages would most likely be sufficient. However, further filtering on top of this will most likely decrease the amount of wrong output.

7.12. Lecture-Data Evaluation

We added the same filtering as for the 2-language setup, ignoring non-available languages which improved the error by about 0.5 % for the tree-net, giving an overall error of 0.175 instead of the 0.179. See tab. 7.11

Table 7.11.: Results of the 6-layer tree-structure evaluated on lecture data (LD) DEV samples.

Filter	**Error**
Bare Net	0.179
3L-Filter	0.175

8. Conclusion

The aim of this thesis was to present an approach to Language Identification (LID) in an online-environment based on Neural Networks. We evaluated our results with three different corpora: The Euronews 2014 corpus, the Lecture Data corpus as collected from the KIT's lecture recordings and talks at Interact25, DGA and a small corpus we collected of European Parliament speeches with their simultaneous translations.

Overall, we have found a feasible approach to Language Identification (LID) in an online-environment. Our setup included a DNBF net to preprocess stacked AFs. The extracted BNFs were stacked with a context and then fed into our LID DNN. We tried DNNs in different setups and net layouts. In general, we can conclude that DNNs are a feasible solution to the LID problem. On our Euronews corpus, for samples longer than 500ms, we achieved an adjusted error rate of 16 % for the tree-net structure with 6 layers that performed the best. This means usage of the net in an online-environment like the lecture translator is feasible.

In a further step we also tried out different net structures on our two other copora: the European Parliament speeches and Lecture Recordings. It was confirmed that the tree-net structure appears to fare the best to identify languages. We also found, that even with a minimal training corpus, as in the case of European Parliament, DNNs are still a feasible classification mechanism, featuring an error of (on samples of any length), only 35 % for a corpus of only 1.5h per language.

Afterwards we evaluated different filtering approaches to be able to further smooth out our LID net output in an online-environment. Overall a counting filter, sequence and gauss filter appear to produce the best results. We defined our own metric, the Out-of-Language Filter as to rate the "noisiness" reduction of a filter. The counting filter features a relative improvement of the Out-Of-Language-Error of 97 % compared to not using a filter. However it would increase delay to 1.5 seconds, which might be considered too big in an online-setting. The sequence filter with a much smaller delay would still future a big improvement however and seems the best choice for the use in an online-environment.

We conclude, that the usage of DNNs to recognize language proved succesfull. As to a possible implementation in the lecture trainslator: the best results were achieved by using a combination of the lecture data and Euronews corpus (section 6.5), as this provided the most robust data. However it is likely the results of a lecture data-only net would prove to be better with a bigger train corpus. Overall, from the results in this thesis, the usage of the concatenated corpus and use of a selection/combination of our post-processing filters to further smooth data, seems to fare the best. A result of an error rate of only 0.065 would certainly be small enough to prove helpful in improving the Lecture Translator in a multilingual environment.

8.1. Future Work

Future work to further improve the results and findings of this thesis could include:

RNNs have recently outperformed DNNs in Language Identification [**?**], so further experiments with the presented pre- /post-processing setup and RNNs instead of DNNs could improve upon our results.

While our findings in regards to the network architecture and setup have been shown to be generic enough to be applicable to three different data corpora, further work should be done to confirm especially the post-processing findings on other (larger) corpora.

Post-Processing in an online-environment has been found to be a non-trivial problem, as can be seen on our two filter metrics diverging. As post-processing is often not the content of other works, future work could focus on finding a suitable online evaluation setup of LID nets, including the post-processing approaches presented in this thesis.

Bibliography

[AAB+16] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016.

[bAO14] N. bt Aripin and M. B. Othman. Voice control of home appliances using android. In *2014 Electrical Power, Electronics, Communicatons, Control and Informatics Seminar (EECCIS)*, pages 142–146, Aug 2014.

[BBB+11] James Bergstra, Frédéric Bastien, Olivier Breuleux, Pascal Lamblin, Razvan Pascanu, Olivier Delalleau, Guillaume Desjardins, David Warde-Farley, Ian Goodfellow, Arnaud Bergeron, et al. Theano: Deep learning on gpus with python. In *NIPS 2011, BigLearning Workshop, Granada, Spain*, volume 3. Citeseer, 2011.

[CM05] Barry R. Chiswick and Paul W. Miller. Linguistic distance: A quantitative measure of the distance between english and other languages. *Journal of Multilingual and Multicultural Development*, 26(1):1–11, 2005.

[Con96] Linguistic Data Consortium. Callfriend corpus, 1996.

[CW08] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[DEGP+12] Luis Fernando D'haro Enríquez, Ondřej Glembek, Oldřich Plchot, Pavel Matějka, Mehdi Soufifar, Ricardo de Córdoba Herralde, and Jan Černockỳ. Phonotactic language recognition using i-vectors and phoneme posteriogram counts. 2012.

[DL08] Yan Deng and Jia Liu. Automatic language identification using support vector machines and phonetic n-gram. In *2008 International Conference on Audio, Language and Image Processing*, pages 71–74, July 2008.

[FHBM08] A.M. Franz, M.H. Henzinger, S. Brin, and B.C. Milch. Voice interface for a search engine, April 29 2008. US Patent 7,366,668.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[Geh12] Jonas Gehring. *Training deep neural networks for bottleneck feature extraction*. 2012. Karlsruhe, KIT, Pittsburgh, Carnegie Mellon Univ., Interactive Systems Laboratories, Masterarbeit, 2012.

[Gre14] Roberto Gretter. Euronews: a multilingual benchmark for asr and lid. In *INTERSPEECH*, pages 1603–1607, 2014.

[HDY+12] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[HN04] Simon Haykin and Neural Network. A comprehensive foundation. *Neural Networks*, 2(2004):41, 2004.

[HSW12] M. Heck, S. Stüker, and A. Waibel. A hybrid phonotactic language identification system with an svm back-end for simultaneous lecture translation. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4857–4860, March 2012.

[LGTB97] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113, 1997.

[LHE08] Kornel Laskowski, Mattias Heldner, and Jens Edlund. The fundamental frequency variation spectrum. *Proceedings of FONETIK 2008*, pages 29–32, 2008.

[LMGDP+14] I. Lopez-Moreno, J. Gonzalez-Dominguez, O. Plchot, D. Martinez, J. Gonzalez-Rodriguez, and P. Moreno. Automatic language identification using deep neural networks. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5337–5341, May 2014.

[LML07] H. Li, B. Ma, and C. H. Lee. A vector space modeling approach to spoken language identification. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(1):271–284, Jan 2007.

[LML13] H. Li, B. Ma, and K. A. Lee. Spoken language recognition: From fundamentals to practice. *Proceedings of the IEEE*, 101(5):1136–1159, May 2013.

[LRY05] M. Leena, K. Srinivasa Rao, and B. Yegnanarayana. Neural network classifiers for language identification using phonotactic and prosodic features. In *Proceedings of 2005 International Conference on Intelligent Sensing and Information Processing, 2005.*, pages 404–408, Jan 2005.

[LWL+97] Alon Lavie, Alex Waibel, Lori Levin, Michael Finke, Donna Gates, Marsal Gavalda, Torsten Zeppenfeld, and Puming Zhan. Janus-iii: Speech-to-speech translation in multiple languages. In *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*, volume 1, pages 99–102. IEEE, 1997.

[MCO92] Yeshwant K Muthusamy, Ronald A Cole, and Beatrice T Oshika. The ogi multi-language telephone speech corpus. In *ICSLP*, volume 92, pages 895–898, 1992.

[Men13] Xiangrui Meng. Scalable simple random sampling and stratified sampling. In *ICML (3)*, pages 531–539, 2013.

[ML06] Bin Ma and Haizhou Li. A comparative study of four language identification systems. *Computational Linguistics and Chinese Language Processing*, 11(2):159–182, 2006.

[MNN+16] Markus Müller, Thai Son Nguyen, Jan Niehues, Eunah Cho, Bastian Krüger, Thanh-Le Ha, Kevin Kilgour, Matthias Sperber, Mohammed Mediani, Sebastian Stüker, and Alex Waibel. Lecture translator - speech translation framework for simultaneous lecture translation. In *Proceedings of the Demonstrations Session, NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 82–86, 2016.

[MSW+13] Florian Metze, Zaid AW Sheikh, Alex Waibel, Jonas Gehring, Kevin Kilgour, Quoc Bao Nguyen, and Van Huy Nguyen. Models of tone for tonal and non-tonal languages. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 261–266. IEEE, 2013.

[MSW16] Markus Müller, Sebastian Stüker, and Alex Waibel. Language adaptive dnns for improved low resource speech recognition. In *Proceedings of the 17th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, San Francisco, USA, September 8-12 2016.

[MY04] Leena Mary and B. Yegnanarayana. Autoassociative neural network models for language identification. In *International Conference on Intelligent Sensing and Information Processing, 2004. Proceedings of*, pages 317–320, 2004.

[MZN+14] Pavel Matejka, Le Zhang, Tim Ng, Harish Sri Mallidi, Ondrej Glembek, Jeff Ma, and Bing Zhang. Neural network bottleneck features for language identification. *Proc. IEEE Odyssey*, pages 299–304, 2014.

[Nie15] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. `http://neuralnetworksanddeeplearning.com`.

[Sch99] Kjell Schubert. Grundfrequenzverfolgung und deren anwendung in der spracherkennung. *Master's thesis, Universität Karlsruhe (TH), Germany*, pages 29–32, 1999.

[SHJ+15] Yan Song, Xinhai Hong, Bing Jiang, Ruilian Cui, Ian Vince McLoughlin, and Lirong Dai. Deep bottleneck network based i-vector representation for

language identification. 2015.

[SJB+13] Y. Song, B. Jiang, Y. Bao, S. Wei, and L. R. Dai. I-vector representation based on bottleneck features for language identification. *Electronics Letters*, 49(24):1569–1570, November 2013.

[SKM+12] Sebastian Stüker, Florian Kraft, Christian Mohr, Teresa Herrmann, Eunah Cho, and Alex Waibel. The kit lecture corpus for speech translation. In *LREC*, pages 3409–3414, 2012.

[TCSK+02] Pedro A Torres-Carrasquillo, Elliot Singer, Mary A Kohler, Richard J Greene, Douglas A Reynolds, and John R Deller Jr. Approaches to language identification using gaussian mixture models and shifted delta cepstral features. In *Interspeech*, 2002.

[VMH+05] David Vilar, Evgeny Matusov, Saša Hasan, Richard Zens, and Hermann Ney. Statistical machine translation of european parliamentary speeches. In *Proceedings of MT Summit X*, pages 259–266, 2005.

[Wer90] P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, Oct 1990.

[Z+96] Marc A Zissman et al. Comparison of four approaches to automatic language identification of telephone speech. *IEEE Transactions on speech and audio processing*, 4(1):31, 1996.

[ZB01] Marc A Zissman and Kay M Berkling. Automatic language identification. *Speech Communication*, 35(1):115–124, 2001.

A. Appendix

In this Appendix we present images, data that did not make it into the main bodz, complete source code listings as well as a glossary at the end.

A.1. Detailed Error rates for Filters

Table A.1.: Results of the best net structure (tree-structure 6-layer) evaluated on the Euronews Development Set with the Advanced small Filter.

Language	**Error (total)**
Arabian	0.365
German	0.303
Spanish	0.321
French	0.304
Italian	0.327
Polish	0.270
Portuguese	0.293
Russian	0.316
Turkish	0.338
English	0.265
Overall	0.313

Table A.2.: Results of the best net structure (tree-structure 6-layer) evaluated on the Euronews Development Set with the Difference Filter.

Language	**Error (total)**
Arabian	0.367
German	0.305
Spanish	0.327
French	0.306
Italian	0.332
Polish	0.276
Portuguese	0.299
Russian	0.323
Turkish	0.341
English	0.269
Overall	0.317

Table A.3.: Results of the best net structure (tree-structure 6-layer) evaluated on the Euronews Development Set with the Gaussian Filter (WS 15).

Language	**Error (total)**
Arabian	0.366
German	0.303
Spanish	0.321
French	0.303
Italian	0.325
Polish	0.271
Portuguese	0.292
Russian	0.315
Turkish	0.338
English	0.265
Overall	0.313

Table A.4.: Results of the best net structure (tree-structure 6-layer) evaluated on the Euronews Development Set with the Counting Filter (WS 100).

Language	**Error (total)**
Arabian	0.362
German	0.300
Spanish	0.319
French	0.302
Italian	0.330
Polish	0.270
Portuguese	0.294
Russian	0.317
Turkish	0.340
English	0.270
Overall	0.313

Table A.5.: Results of the best net structure (tree-structure 6-layer) evaluated on the Euronews Development Set with the Sequence Filter (WS 10)

Language	**Error (total)**
Arabian	0.401
German	0.359
Spanish	0.388
French	0.368
Italian	0.427
Polish	0.346
Portuguese	0.394
Russian	0.422
Turkish	0.445
English	0.370
Overall	0.393

Table A.6.: Results of the best net structure (tree-structure 6-layer) evaluated on the Euronews Development Set with the Speech/Noise Filter

Language	**Error (total)**
Arabian	0.366
German	0.301
Spanish	0.320
French	0.304
Italian	0.330
Polish	0.268
Portuguese	0.295
Russian	0.315
Turkish	0.336
English	0.272
Overall	0.313

A.2. Complete Source Code Listings

As mentioned in ch. 7, we are listing the complete source code for all the different smoothing algorithms tried here, as well as the full source code for the feature description.

```
puts "start featDesc"

#------------------speech detection-------------------
$fes    adc2pow          POWER    ADC      16ms #Extract sound power
$fes    alog             POWER    POWER    1 4 #Log on sound power
$fes    filter           POWER    POWER    {-2 {1 2 3 2 1}} #Weighted Average
     with context of 2
$fes    normalize        POWER    POWER    -min -0.1 -max 0.5 #Normalize
$fes    thresh           SPEECH   POWER    1.0 0 upper #> 0 -> Speech
$fes    thresh           SPEECH   SPEECH   0    0 lower #< 0 -> No Speech

#----------------- mel filter bank------------------
$fes    spectrum         FFT0              ADC              16ms
set WARP 1.0

### computing log-mel
#Help Check to not allocate mel filterbank coefficient matrix twice (
    will lead to tcl error)
if { [llength [objects FBMatrix matrixMEL]] != 1} {
        set melN 40
        set points [$fes:FFT0 configure -coeffN]
        set rate   [expr 1000 * [$fes:FFT0 configure -samplingRate]]
        [FBMatrix matrixMEL] mel -N $melN -p $points -rate $rate
}

$fes    VTLN             FFT              FFT0             $WARP -mod lin
    -edge 0.8
$fes    filterbank       MEL              FFT              matrixMEL
$fes    log              lMEL             MEL              1.0 1.0 #Log-Mel

 # Computing Tonal feature
# pitch
$fes pitch               PITCH  ADC  16ms
$fes delta               dPITCH1 PITCH -delta 1
$fes delta               ddPITCH1 dPITCH1 -delta 1
$fes delta               dPITCH2 PITCH -delta 2
$fes delta               ddPITCH2 dPITCH2 -delta 2
$fes delta               dPITCH3 PITCH -delta 3
$fes delta               ddPITCH3 dPITCH3 -delta 3
$fes merge               P16 PITCH dPITCH1 ddPITCH1 dPITCH2 ddPITCH2
    dPITCH3 ddPITCH3

# Calculating FFV
$fes intonation          FFV  ADCffv 32ms -tint 11ms -text 9ms -tsep 14ms
     # On 32 ms Windows!
$fes merge               TONE FFV  P16

$fes    merge            mergedFEAT           lMEL       TONE
```

```
44 $fes    meansub          FEAT                    mergedFEAT       -a 2 -weight
       SPEECH #Weigh with the Speech feature.
45
46 $fes    adjacent         FEATSPR                 FEAT             -delta 6 #
       Stack Context of 6 frames
```

Listing A.1: The feature description as used by our pre-DBNF-preprocessing

```
proc filter{} {
    global totalE
    #setting up variables
     for {set i 0} {$i < 10} {incr i} {
        set totalM($i) 0
    }
    set currentOutput -1
    #Going through whole sample frame by frame in output layer of nn
       called nnBNF-> can be changed to work on continuously incoming
       data easily
    for {set i 0} { $i < [featureSetLID frameN nnBNF]} {incr i} {
        #we find the current output of the net
        set maxFrame [lindex [lsort -decreasing -real [featureSetLID
           frame nnBNF $i]] 0]
        set maxFrameID [lsearch -real [featureSetLID frame nnBNF $i]
           $maxFrame]
        set currentOutput $maxFrameID
        #setting total classification amounts for current sample
        if {$currentOutput != -1} {
             set totalM($currentOutput) [expr {[set totalM(
                $currentOutput)] + 1}]
        }
    }
    set maxOverall -1
    set maxID -1
    set totalN 0
    #Now get the output for the whole sample (smoothing over entire
       sample)
    for {set i 0} {$i < 10} {incr i} {
        if {[set totalM($i)] > $maxOverall} {
            set maxOverall [set totalM($i)]
            set maxID $i
        }
        set totalN [expr $totalN + [set totalM($i)]]
    }
    puts "totalN $totalN"
    puts "maxOverall $maxOverall"
    set wrongN [expr $totalN - $maxOverall]
    puts "wrongN $wrongN"
    set totalE [expr {(double($wrongN) / $totalN)}]
    #TotalE is the OLE of the sample
    puts "totalE $totalE"

    #print out the total classification for the entire sample
    puts -nonewline "Overall we have classified as: "
    #help function to print language name not id.0
    puts [getName $maxID]
    return $maxID
}
```

Listing A.2: Evaluation setup to count outputs per sample and count correctness(return value)/OLE (totalE)

```
proc filter {} {
    global totalE
    for {set i 0} {$i < 10} {incr i} {
    set total($i) 0
    }
    set lastFrameID -1
    set counter 0
    set currentOutput -1
    for {set i 0} { $i < [featureSetLID frameN nnBNF]} {incr i} {
        set maxFrame [lindex [lsort -decreasing -real [featureSetLID
            frame nnBNF $i]] 0]
        set maxFrameID [lsearch -real [featureSetLID frame nnBNF $i]
            $maxFrame]
        if {$maxFrameID != $lastFrameID} {
            set lastFrameID $maxFrameID
            set counter 0
        } elseif {$maxFrame >= 0.61} {
            incr counter
            if {$counter >= 5} {
                set currentOutput $maxFrameID
            }
        }
        if {$currentOutput != -1} {
             incr total($currentOutput)
        }
    }
    set maxOverall -1
    set maxID -1
    set totalN 0
    for {set i 0} {$i < 10} {incr i} {
        if {$total($i) > $maxOverall} {
             set maxOverall $total($i)
             set maxID $i
        }
        set totalN [expr $totalN + [set totalM($i)]]
    }
    set wrongN [expr $totalN - $maxOverall]
    set totalE [expr {(double($wrongN) / $totalN)}]
    puts -nonewline "Overall we have classified as: "
    puts [getName $maxID]
}
```

Listing A.3: Basic (first try) Filter employed to smooth/improve output

```
proc filter {} {
    global totalE

    for {set i 0} {$i < 10} {incr i} {
        set totalM($i) 0
    }
    set lastFrameID -1
    set counter 0
    set currentOutput -1
    for {set i 0} { $i < [featureSetLID frameN nnFILTER]} {incr i} {
        set maxFrame [lindex [lsort -decreasing -real [featureSetLID
            frame nnFILTER $i]] 0]
        set maxFrameID [lsearch -real [featureSetLID frame nnFILTER $i]
            $maxFrame]
        set currentOutput $maxFrameID
        if {$currentOutput != -1} {

            set totalM($currentOutput) [expr {[set totalM($currentOutput
                )] + 1}]
        }
    }
    set maxOverall -1
    set maxID -1
    set totalN 0
    for {set i 0} {$i < 10} {incr i} {
        if {[set totalM($i)] > $maxOverall} {
            set maxOverall [set totalM($i)]
            set maxID $i
        }
        set totalN [expr $totalN + [set totalM($i)]]
    }
    puts "totalN $totalN"
    puts "maxOverall $maxOverall"
    set wrongN [expr $totalN - $maxOverall]
    puts "wrongN $wrongN"
    set totalE [expr {(double($wrongN) / $totalN)}]

    puts "totalE $totalE"
    puts -nonewline "Overall we have classified as: "
    puts [getName $maxID]
    return $maxID
}
```

Listing A.4: Advanced FILTER employed to smooth/improve output

```
proc filter {} {
    global totalE
    for {set i 0} {$i < 10} {incr i} {
    set totalM($i) 0
    }
    set lastFrameID -1
    set counter 0
    set currentOutput -1
    for {set i 0} { $i < [featureSetLID frameN nnBNF]} {incr i} {
    set maxFrame [lindex [lsort -decreasing -real [featureSetLID frame
        nnBNF $i]] 0]
    set max2Frame [lindex [lsort -decreasing -real [featureSetLID frame
        nnBNF $i]] 1]
    set maxFrameID [lsearch -real [featureSetLID frame nnBNF $i]
        $maxFrame]
    set max2FrameID [lsearch -real [featureSetLID frame nnBNF $i]
        $max2Frame]
    if {$maxFrameID != $lastFrameID} {
        set lastFrameID $maxFrameID
        set counter 0
    } elseif {[expr {$maxFrame - $max2Frame >= 0.01}]} {
        incr counter
        if {$counter >= 5} {
            set currentOutput $maxFrameID
        }
    }
    if {$currentOutput != -1} {
                set totalM($currentOutput) [expr {[set totalM(
                    $currentOutput)] + 1}]
    }
    }
   set maxOverall -1
    set maxID -1
    set totalN 0
    for {set i 0} {$i < 10} {incr i} {
        if {[set totalM($i)] > $maxOverall} {
            set maxOverall [set totalM($i)]
            set maxID $i
        }
        set totalN [expr $totalN + [set totalM($i)]]

    }
    set wrongN [expr $totalN - $maxOverall]
    set totalE [expr {(double($wrongN) / $totalN)}]
    puts -nonewline "Overall we have classified as: "
    puts [getName $maxID]
    return $maxID
}
```

Listing A.5: Difference Filter employed to smooth/improve output

```
proc filter {} {
    global totalE
    for {set i 0} {$i < 10} {incr i} {
        set totalM($i) 0
        set count($i) 0
    }
    set lastFrameID -1
    set counter 0
    set currentOutput -1
    for {set i 0} { $i < [featureSetLID frameN nnBNF]} {incr i} {
        set maxFrame [lindex [lsort -decreasing -real [featureSetLID
            frame nnBNF $i]] 0]
        set maxFrameID [lsearch -real [featureSetLID frame nnBNF $i]
            $maxFrame]
        for {set j 0} {$j < 10 && $i < [featureSetLID frameN nnBNF]} {
            incr j} {
            set maxFrame [lindex [lsort -decreasing -real [featureSetLID
                frame nnBNF $i]] 0]
            set maxFrameID [lsearch -real [featureSetLID frame nnBNF $i]
                $maxFrame]
            set count($maxFrameID) [expr {[set count($maxFrameID)] + 1}]
            incr i
        }
       set maxAvg -1
       set maxID -1
       for {set k 0} {$k < 10} {incr k} {
          if {[set count($k)] > $maxAvg} {
              set maxAvg [set count($k)]
              set maxID $k
         }
       }
      set currentOutput $maxID
      if {$currentOutput != -1} {
              set totalM($currentOutput) [expr {[set totalM(
                  $currentOutput)] + 1}]
      }
    }
    set maxOverall -1
    set maxID -1
    set totalN 0
    for {set i 0} {$i < 10} {incr i} {
        if {[set totalM($i)] > $maxOverall} {
            set maxOverall [set totalM($i)]
            set maxID $i
        }
        set totalN [expr $totalN + [set totalM($i)]]

    }
    set wrongN [expr $totalN - $maxOverall]
    set totalE [expr {(double($wrongN) / $totalN)}]
    puts -nonewline "Overall we have classified as: "
    puts [getName $maxID]
    return $maxID
```

```
48 }
```

Listing A.6: Counting Filter employed to smooth/improve output

```
proc filter {} {
    global totalE
    for {set i 0} {$i < 10} {incr i} {
        set totalM($i) 0
    }
    set lastFrameID -1
    set counter 0
    set currentOutput -1
    for {set i 0} { $i < [featureSetLID frameN nnBNF]} {incr i} {
        set maxFrame [lindex [lsort -decreasing -real [featureSetLID
            frame nnBNF $i]] 0]
        set maxFrameID [lsearch -real [featureSetLID frame nnBNF $i]
            $maxFrame]
        set currentOutput $maxFrameID
        if {$currentOutput != -1 && [featureSetLID frame SPEECH $i] == "
            1.000000e+00"} {
            set totalM($currentOutput) [expr {[set totalM($currentOutput
                )] + 1}]
            set lastFrameID $currentOutput
        } elseif {$lastFrameID != -1} {
            set totalM($lastFrameID) [expr {[set totalM($lastFrameID)] +
                1}]
        }
    }

    set maxOverall -1
    set maxID -1
    set totalN 0
    for {set i 0} {$i < 10} {incr i} {
        if {[set totalM($i)] > $maxOverall} {
            set maxOverall [set totalM($i)]
            set maxID $i
        }
        set totalN [expr $totalN + [set totalM($i)]]

    }
    set wrongN [expr $totalN - $maxOverall]
    set totalE [expr {(double($wrongN) / $totalN)}]
    puts -nonewline "Overall we have classified as: "
    puts [getName $maxID]
    puts -nonewline "Length of sample: "
    puts [featureSetLID frameN nnBNF]
    return $maxID
}
```

Listing A.7: Speech Filter employed to smooth/improve output

```
proc filter {} {
    global totalE

    for {set i 0} {$i < 10} {incr i} {
        set totalM($i) 0
    }
    set lastFrameID -1
    set counter 0
    set currentOutput -1
    for {set i 0} { $i < [featureSetLID frameN nnBNF]} {incr i} {
    set maxFrame [lindex [lsort -decreasing -real [featureSetLID frame
        nnBNF $i]] 0]
    set maxFrameID [lsearch -real [featureSetLID frame nnBNF $i]
        $maxFrame]
    if {$maxFrameID != $lastFrameID} {
        if {($maxFrameID == 1 || $maxFrameID == 9)} {
         set lastFrameID $maxFrameID
         set currentOutput $maxFrameID
      } elseif {$lastFrameID  == 1 || $lastFrameID == 9} {
         set currentOutput $lastFrameID
        }
    } else {
        set currentOutput $lastFrameID
    }

    if {$currentOutput != -1 && ($currentOutput == 1 || $currentOutput
        == 9)}  {
               set totalM($currentOutput) [expr {[set totalM(
                   $currentOutput)] + 1}]
         }
    }
    set maxOverall -1
    set maxID -1
    set totalN 0
    for {set i 0} {$i < 10} {incr i} {
        if {[set totalM($i)] > $maxOverall} {
            set maxOverall [set totalM($i)]
            set maxID $i
        }
        set totalN [expr $totalN + [set totalM($i)]]
    }
    puts "totalN $totalN"
    puts "maxOverall $maxOverall"
    set wrongN [expr $totalN - $maxOverall]
    puts "wrongN $wrongN"
    if {$totalN != 0} {
        set totalE [expr {(double($wrongN) / $totalN)}]
    } else {
        set totalE 1
    }
    puts "totalE $totalE"
    puts -nonewline "Overall we have classified as: "
    puts [getName $maxID]
```

```
50      puts -nonewline "Length of sample: "
51      puts [featureSetLID frameN nnBNF]
52      return $maxID
53  }
```

Listing A.8: 2-Language Filter (For DE/EN) employed to smooth/improve output

```
proc getMax {} {
    global totalE

    for {set i 0} {$i < 10} {incr i} {
        set totalM($i) 0
        set countM($i) 0
    }
    set lastFrameID -1
    set counter 0
    set currentOutput -1
    for {set i 0} { $i < [featureSetLID frameN nnBNF]} {set i [expr $i +
        100]} {
        set curID -1
        set curCount 0
        set currentOutput -1
        for {set j 0} {[expr $i + $j] < [featureSetLID frameN nnBNF] &&
            $j < 100} {incr j} {
            set maxFrame [lindex [lsort -decreasing -real [featureSetLID
                frame nnBNF [expr $i + $j]]] 0]
            set maxFrameID [lsearch -real [featureSetLID frame nnBNF [
                expr $i + $j]] $maxFrame]
            if {$maxFrameID != $curID} {
                set curID $maxFrameID
                set curCount 1
            } elseif {$maxFrameID != -1} {
                incr curCount
            }
            if {$curCount > [set countM($curID)]} {
                set countM($curID) $curCount
            }
        }

        set currentMax -1
        for {set k 0} {$k < 10} {incr k}    {
            if {[set countM($k)] > $currentMax} {
                set currentMax [set countM($k)]
                set currentOutput $k
            }
        }
        if {$currentOutput != -1}  {
            set totalM($currentOutput) [expr {[set totalM($currentOutput
                )] + 1}]
        }
    }

    set maxOverall -1
    set maxID -1
    set totalN 0
    for {set i 0} {$i < 10} {incr i} {
        if {[set totalM($i)] > $maxOverall} {
            set maxOverall [set totalM($i)]
            set maxID $i
        }
```

```
49          set totalN [expr $totalN + [set totalM($i)]]
50      }
51      puts "totalN $totalN"
52      puts "maxOverall $maxOverall"
53      set wrongN [expr $totalN - $maxOverall]
54      puts "wrongN $wrongN"
55      set totalE [expr {(double($wrongN) / $totalN)}]
56
57      puts "totalE $totalE"
58
59      puts -nonewline "Overall we have classified as: "
60      puts [getName $maxID]
61      puts -nonewline "Length of sample: "
62      puts [featureSetLID frameN nnBNF]
63      return $maxID
64  }
```

Listing A.9: Sequence Filter employed to smooth/improve output

```
#Windowsize definition
set ws 5
set sigma [expr sqrt($ws/(2*[Pi]))]
#calculate gauss kernel
for {set k 0} {$k <= $ws} {incr k} {
    set gauss($k) [expr (1/(sqrt(2*[Pi])*$sigma)) * ([E] ** - (( $k ** 2)/ (2 * $sigma ** 2)))]
}
#For each frame
for {set i 0} { $i < [featureSetLID frameN nnBNF]} {incr i} {
    for {set j 0} {$j < 10} {incr j} {
        set values($j) [lindex [featureSetLID frame nnBNF $i] $j]
        set curValue 0
        #go from $i - $ws to $i + $ws
        for {set k [expr -$ws]} {$k < $ws} {incr k} {
            #Only use the current value if it actually exists
            if {[expr $i + $k] >= 0 && [expr $i + $k] < [
                featureSetLID frameN nnBNF]} {
                set cur [lindex [featureSetLID frame nnBNF [expr $i + $k]] $j]
                #get correct index for gauss kernel, as we only calculated once above for symmetric function
                if {$k < 0} {
                    set curIdx [expr -$k]
                } else {
                    set curIdx $k
                }
                #running sum of all previous values in the window convoluted with the gaussian kernel.
                set curValue [expr $curValue + $cur * [set gauss($curIdx)]]
            }
        }
        #save the value for current language coefficient
        set values($j) $curValue
    }
    #Find maximum on smoothed values
    set max -1
    set maxID -1
    for {set j 0} {$j < 10} {incr j} {
        if {[set values($j)] > $max} {
            set max [set values($j)]
            set maxID $j
        }
    }
    #Set the output as the Language with Max. coefficient.
    set currentOutput $maxID
}
```

Listing A.10: Gaussian Smoothing Filter as implemented in tcl/tk for the JRTk