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

# Zusammenfassung

# Contents

**Abstract** i

**Zusammenfassung** iii

**1. Introduction** 1
- 1.1. Motivation . . . 1
- 1.2. Overview . . . 2

**2. Related Work** 3
- 2.1. Historically . . . 3
- 2.2. Recent Approaches . . . 3
- 2.3. Similar Approaches . . . 4

**3. Fundamentals** 5
- 3.1. Theory of Language Identification . . . 5
- 3.2. Janus Recognition Toolkit (jrtk) . . . 5
- 3.3. Neural Networks . . . 6
  - 3.3.1. Neural Networks . . . 6
  - 3.3.2. Artificial Neuron . . . 6
  - 3.3.3. General Network Setup . . . 7
  - 3.3.4. Network Types . . . 8
  - 3.3.5. Learning . . . 9

**4. Experimental Setup** 13
- 4.1. Language Identity Neural Networks . . . 13
- 4.2. Tooling . . . 14
- 4.3. Euronews 2014 . . . 14
- 4.4. Lecture Data . . . 16
- 4.5. European Parliament . . . 17

**5. Feature Preprocessing** 19
- 5.1. Feature Retrieval . . . 19
- 5.2. Feature Description . . . 19
- 5.3. ASR BNF network . . . 21
- 5.4. Changing the Input Features . . . 22

**6. LID Network Structure** 23
- 6.1. Basic Setup . . . 23

6.2. Improving Network Layout . . . . . 23
6.3. Big Euronews Corpus . . . . . 25
6.4. Finetuning . . . . . 26
6.5. Combining Lecture Data and Euronews . . . . . 27
6.6. Lecture-Data-Training . . . . . 27

**7. Smoothing and Evaluation** **29**
7.1. Bare net output . . . . . 29
7.2. Basic Test Filter . . . . . 30
7.3. Advanced Test Filter . . . . . 32
7.4. Difference Test Filter . . . . . 32
7.5. Counting Filter . . . . . 33
7.6. Two-Language Setup . . . . . 33
7.7. Gaussian Smoothing Filter . . . . . 34
7.8. Lecture Data . . . . . 36

**8. Conclusion** **39**

**A. Appendix** **45**
A.1. Complete Source Code Listings . . . . . 45

# List of Figures

3.1. A schematic drawing of a Neuron and it's parameters . . . . . 7
3.2. Basic Feed-Forward fully-connected Neural Network. . . . . 8

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. . . . . 14

# List of Tables


4.1. The Euronews corpus speaker breakdown used for most experiments with total utterances length. . . . . 15
4.2. The Euronews corpus breakdown into the three data sets. . . . . 15
4.3. The big Euronews corpus speaker breakdown with total utterances length. 16
4.4. The Lecture Data corpus speaker breakdown with total utterances length. 16

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. . 25
6.2. Results of the different net structures on the Euronews corpus. Frame-based errors on train/validation set. . . . . 26
6.3. Comparison of the big and small Euronews data corpus. Frame-based errors on train/validation set, as well as Sample-based Error. . . . . 26
6.4. Results of the fine-tuning attempts of the Euronews-trained 6-layer tree-net with lecture data corpus. . . . . 27
6.5. Results of the fine-tuning attempts of the Euronews-trained 6-layer tree-net with lecture data corpus. . . . . 28

7.1. Results of the best net structure (tree-structure 6-layer) evaluated on the Euronews Development Set. . . . . 31
7.2. Results of the best net structure (tree-structure 6-layer) evaluated on the Euronews Development Set with the Basic Filter. . . . . 32
7.3. Results of the best net structure (tree-structure 6-layer) evaluated on only 2 Languages from the Euronews Development Set. . . . . 33
7.4. Results of the 6-layer tree-structure evaluated with all the tried post-filtering approaches. . . . . 36

# 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 Network's, a Machine Learning approach to classification tasks employed greatly throughout all sciences and especially in computer science and tasks concerned with the processing of spoken speech. This thesis tries to find a low-latency, fast, or "online", approach to Language Identification using the classification method of neural networks.

The work done in this thesis uses the Janus Recognition Toolkit (jrtk)[1], an Automatic Speech Recognition toolkit developed in joint cooperation by the KIT and CMU. The jrtk offers a tcl/tk[2] script-based environment for the development of Automatic Speech Recognition 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

Automatic Speech Recognition (ASR) is used in many applications and devices today, especially in the rise of handheld mobile devices like smart-phones and tablets. It has progressed quickly in the last five years and has found commercial success. Famous examples include Google[3]'s "Ok, Google" and Apple[4]'s 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[5], especially relevant for this thesis in the realm of Lecture Translation[MNN+16] .

The task of Language Identification can be applied in all of those fields, as Automatic Speech Recognition is mostly trained on one language and therefore requires a totally different setup of classifiers per language, making a manual change of language previous to recognition necessary. Robust and low-latency language identification would eliminate the need for this.

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

[1] Janus Recognition Toolkit: `http://isl.anthropomatik.kit.edu/cmu-kit/english/1406.php`
[2] Tcl/tk: `https://www.tcl.tk/`
[3] Google: `www.google.com`
[4] Apple: `www.apple.com`
[5] IWSLT: `iwslt.org`

Translation are employed and are being actively developed in the TC-STAR inititative[6], 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[7] as the system trained was implemented for it. We believe our results are general enough to be transferable to other applications with small implementation-specific changes.

## 1.2. Overview

The following chapter gives an introductory view of the applications of Language Identification, and introduces the tasks this thesis tries to solve. Afterwards we give preliminary theoretical explanations and definitions, including an introduction to Neural Networks in Sec. 3.3. The next chapter describes the experimental setup used in this thesis, including the data corpusses and feature preprocessing done on raw audio files.

Chapters 4 and 5 describe our results that were accomplished by trying out different Network Structures and geometries in chapter 4 as well as different smoothing mechanisms on top of the direct neuronal output layer of the network in chapter 5. This is followed by the final summary of our work and an outlook onto future work improving upon these results.

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

[7]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 their 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 PRR 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, however.

## 2.2. Recent Approaches

Li et al. in [LML07], 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. By determining the top most occurring words in a language, they rely on these statistics to distance languages from each other, then applying a language-dependent language model on top to classify. A similar approach is proven to be successful with support vector machines (SVM) in [DL08], [HSW12] 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. Leena et al. in [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].

## 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 as the focus lies on for us.

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

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, as to make understanding of the following chapters easier.

## 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 frame 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 optimal 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, or a tokenizer or similar approaches before trying to identify the language, we assume that each speech sequence can be segmented into a sequence $\Upsilon$ 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 $\Upsilon$ 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 $\Upsilon^O$:

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

Combining 3.2 and 3.3 and considering all possible $\Upsilon$ instead of the best hypothesis $\Upsilon^O$ 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 [LWL+97]. 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 it's 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 and very successfully 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 make the understanding of later chapters easier for the reader. The information in this section is based mostly on [HN04], [GBC16] and [Nie15]

### 3.3.1. Neural Networks

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 and 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" and 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 $\Theta$ 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

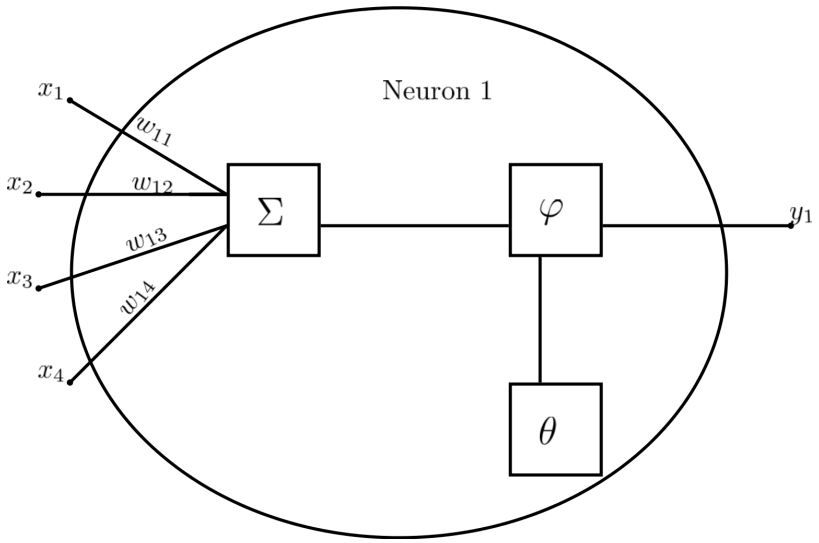
An artificial neuron, or perceptron in its most basic form, is a mathematical function that consists of four parameters that can be adjusted independently from each other:

- $w_i$ the input weights for all inputs
- $\Sigma$ the transfer function for summation of the weighted inputs
- $\varphi$ the activation function that calculates the output value $y_k$ basend on the transfer input and the threshold
- $\theta$ 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 it's parameters




### 3.3.3. General Network Setup

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 entering input, 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.

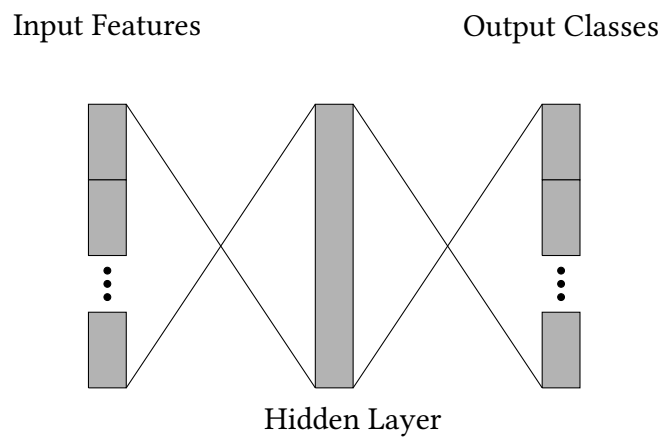



Figure 3.2.: Basic Feed-Forward fully-connected Neural Network.

### 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, in opposition to *Recurrent Neural Networks*, that connections between the neural units are not cyclic.

In such a basic network, the output layer consists of as many neurons as classes that the network is trying to classify against and the neuron with the highest activation after entering input, is the classification output of the net.

#### 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 phenomen 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. For example, after knowing 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 cyclic connections are allowed. This means the network can have temporal behavior, so its

performance changes dynamically over time, when the state of later neurons changes and affects the neurons in previous layers.

### 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 to be 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 $\Theta$ 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 $\Theta$ each run.

**Three basic approaches exist for training a network:**

- *Supervised Training*, where the optimal output for input 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 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 is most commonly, as in this thesis, done using the

#### 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 $\Theta$ 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 and $f(x;\Theta)$ 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 $\eta$ 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 the next setup

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 the same. 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 $2^{2nd}$ 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.

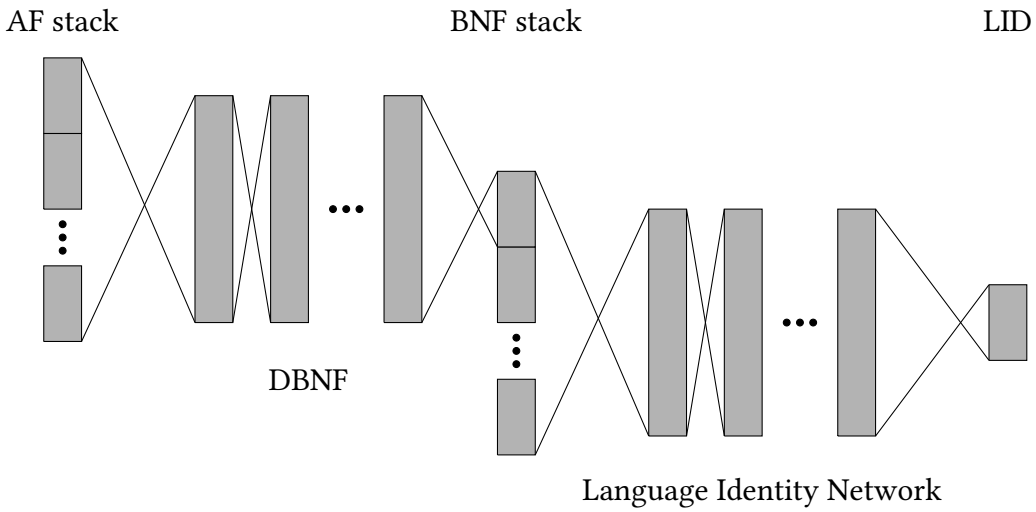



Figure 4.1.: Overview of the network architecture used to extract the Language Identity (LID). The acoustic features (AF) are being pre-processed in a DBNF in order to extract BNFs. These BNFs are being stacked and fed into the second network to extract the LID.

## 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 4.3.

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

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

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

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% for both the development and test set. Table 4.3 shows the split data for the three sets.

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. **??**. The breakdown of the large corpus' data is listed in Table 4.3.

| 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 |

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

## 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 InterACT - 25[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 try out 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.

| 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 |

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

[4]InterACT 25: `http://www.interact25.org/`
[5]DGA: `http://www.defense.gouv.fr/dga`

## 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 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 Lecture-Data- as well as Euronews-trained nets, as well as improve capabilities of the networks with fine-tuning using the Euronews data, 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.

[6] EU-Parliament plenary speeches: `http://www.europarl.europa.eu/ep-live/en/plenary/`
[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 using 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 samplerate of 44.1 kHz. We downsampled the data using soX[1].

**Feature Window Size Discrepancy** As pitch feature were calculated on 16ms windows while intonation was calculated on 32 ms we had to shorten the length of the section the 32ms features are calculated one by 16ms so the number of frames match and the features can be merged. This was done by removing 0.008 s from the beginning and end of the audio file. This was also done to any evaluation audio used with the small Euronews nets.

**Big Euronews Corpus** As the Big Euronews corpus as introduced already had training pfiles existing from previous work [MSW16], we reused them. This meant that the features here were completely based on 32ms windows and no pre-calculation cutting of the audio was necessary.

## 5.2. Feature Description

For the DBNF network as input, a combination of sound power, lMEL, fundamental frequency variation (FFV) [LHE08] and pitch acoustic features [Sch99] was used. Tonal

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

features have shown improvements for DNNs, even with non-tonal languages as English [MSW+13], so they were also incorporated. Lst. 5.1 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 as when the power of the sound in the recording reaches a threshold we consider it speech, otherwise noise. Calculations were done by calculating the Log on the current sound power, calculating the weighted average with a context of 2 frames on each side on this, normalizing the resulting value between -0.1 and 0.5 and then using a threshold of 0 in the normalized spectrum to consider it speech. We do not discard the non-speech marked parts but weigh the combined feature vector less for the total input if the threshold of 0 was not reached.

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: $(PITCH(t), f(t) := PITCH(t+1) - PITCH(t-1), f(t+1) - f(t-1), g(t) := PITCH(t+2) - PITCH(t-2), g(t+2) - g(t-2), h(t) := PITCH(t+3) - PITCH(t-3), h(t+3) - h(t-3))$

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. 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.

**Big Euronews Corpus** The big 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 corpusses we kept the 16ms windows for Power and lMEL, but changed the Pitch to be calculated on 32 ms windows.

```
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
$fes    meansub         FEAT                mergedFEAT      -a 2 -weight
     SPEECH #Weigh with the Speech feature.

$fes    adjacent        FEATSPR             FEAT            -delta 6 #
    Stack Context of 6 frames
```

Listing 5.1: The feature description as used by our pre-DBNF-preprocessing

## 5.3. ASR BNF network

The DBNF use was trained as part of [MSW16]. It is trained using the input features as just presented with a context of 6. The targets are Context-Dependent phoneme states, so a similar setup to how an ASR neural network would look. The network has 6 layers of 1000 neurons each plus a bottleneck layer of only 42 neurons before the final output layer.

The activations of the bottleneck layer are extracted, the context of 11 frames applied and used as the input features for the LID network on which the following chapter will elaborate.

## 5.4. Changing the Input Features

Work in [MSW16] included a comparison of different context spreads as stacked inputs for the LID net of 2,3 and 6. So, as one of the first experiments on the Euronews corpus we increased the spread 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 different net structures, as introduced in the following, with different contexts.

# 6. LID Network Structure

This chapter describes the actual Language Identification Neural Network trained as well as the results of different network/data setups used. Most network experiments in the Network Geometry, meaning the number of hidden layers as well as the layout of the neurons in these layers were tried using the Euronews corpus. Results from this corpus were then transferred over to the other corpusses, 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. Basic Setup

The first experimental net setup consisted of 5 layers of denoising auto-encoders with each 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 2m and 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 we managed to make upon our initial result.

## 6.2. Improving Network Layout

Different experiments were undertaken with the network layout. This includes a 6 hidden layer pre-training as well as changes in the geometry. The differences in the frame-based errors can be seen summarized 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 a 6th hidden layer** Adding another 1000 neuron hidden layer did not lead to an increase in recognition compares to the version using a lower learning rate and rather gave us an decrease of about 2%, meaning a 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

```
'!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.

An 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 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 much 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' \
```

performed not at all featuring an error rate of 90 %, meaning the net is not classifying at all 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.

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 |

## 6.3. Big Euronews Corpus

As introduced in Sec. 4.3, a bigger Euronews data was used to test out our network structure. The big 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.

| Net Structure | Train Error | Validation Error |
|---|---|---|
| Basic (5 layers 1000 Neurons) | 0.226 | 0.285 |
| 6-Layers (1000 Neurons) | 0.236 | 0.276 |
| Reduced Learning Rate | 0.273 | 0.266 |
| Tree Structure (5 layers 1000...200 neurons) | 0.221 | 0.245 |
| Tree Structure (5 layers 1000...200 neurons) and reduced Learning Rate | 0.245 | 0.235 |
| Tree Structure (6 layers 1200...200 neurons) | 0.211 | 0.242 |
| Tree Structure (6 layers 1200...200 neurons) and reduced Learning Rate | 0.251 | 0.238 |
| Tree Structure (10 layers 2000...200 neurons) | 0.899 | 0.910 |
| **Best** | 0.245 | 0.235 |

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

It only improved the per-sample error (as further introduced in the next chapter), by about 4% however, suggesting that a suitable post-net filtering approach and net setup is even more important than the amount of train data, as the data corpus was more than trippled in size but did not improve the results in the same way.

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

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.

## 6.4. Finetuning

As a further experiment, we used the pre-trained nets of the stacked hidden layers of the Euronews net and fine-tuned them with the cross-set of Lecture-Data. This was done by loading net-parameters from pre-training and training the final output layer with 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 fine-tuning) using the cross-set.

For this experiment we used the tree-net 6-layer structure of the Euronews nets, just omitting the output layer, and 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).

| Net Structure | Euronews DE/FR/EN Error | LD Train Error | LD Val. Error | LD Sample Error |
|---|---|---|---|---|
| Tree-net Euronews net w/o fine-tuning | 0.291 | - | - | 0.179 |
| Tree-net Euronews net with fine-tuning | 0.456 | 0.075 | 0.116 | 0.130 |
| Tree-net Euronews net with fine-tuning 2 layers | 0.413 | 0.073 | 0.102 | 0.112 |
| **Change** | **0.121** | - | - | **0.067** |

Table 6.4.: Results of the fine-tuning attempts of the Euronews-trained 6-layer tree-net with lecture data corpus.

A second experiment was conducted, using the same base Euronews net, but adding 2 layers on top for the fine-tuning 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 Fine-tuning compared to the base Euronews net. See Tab. 6.4 for the detailed results.

Since the Lecture Data corpus only has 3 languages, we couldn't evaluate the fine-tuned 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 6.6. This was to be expected just from the amount of train data almost doubling for the 3 lecture data-languages. However the results provided a surprise, in that they did not improve upon lecture-data fine tuning results.

## 6.6. Lecture-Data-Training

After trying fine-tuning 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 tried the same 6-layer setup with and without an adjusted learning rate, and also tried a smaller, 4-layer tree-net with a structure of:

| Net Structure | Euronews DE/FR/EN Error | LD Train Error | LD Val. Error | LD Sample Error |
|---|---|---|---|---|
| Tree-net Euronews net w/o fine-tuning | 0.291 | - | - | 0.179 |
| Concatenated Euronews/LD-trained net | | | | |
| Tree-net Euronews net with fine-tuning | 0.456 | 0.075 | 0.116 | 0.130 |
| Tree-net Euronews net with fine-tuning 2 layers | 0.413 | 0.073 | 0.102 | 0.112 |
| **Change** | **0.121** | **0.002** | **0.014** | **0.018** |

Table 6.5.: Results of the fine-tuning attempts of the Euronews-trained 6-layer tree-net with lecture data corpus.

```
'!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, 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.

Table 6.6 shows a comparison of the different Lecture Data experiments.

# 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 Sec. 5 → ASR BNF extraction Sec. 5.3 → LID network Sec. 6). The following smoothing approaches were applied in an ”online” fashion, meaning it was made sure they can be calculated “on-the-fly” while new data is still coming in. The summarized results are listed in Tab. **??**. We evaluated different net structures, however the results presented here are mostly based on the tree-net 6-layer structure as introduced in the previous chapter. This is not necessarily the best structure for frames, but gave us the best results on sample-basis.

## 7.1. Bare net output

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 Lst. 7.1 for the corresponding tcl/tk code for this basic filtering 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 development 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.

```
proc filter{} {
    #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
```

```
13          #setting total classification amounts for current sample
14          if {$currentOutput != -1} {
15               set totalM($currentOutput) [expr {[set totalM(
                     $currentOutput)] + 1}]
16          }
17      }
18      set maxOverall -1
19      set maxID -1
20      #Now get the output for the whole sample (smoothing over entire
            sample)
21      for {set i 0} {$i < 10} {incr i} {
22          if {[set totalM($i)] > $maxOverall} {
23              set maxOverall [set totalM($i)]
24              set maxID $i
25          }
26      }
27      #print out the total classification for the entire sample
28      puts -nonewline "Overall we have classified as: "
29      #help function to print language name not id.0
30      puts [getName $maxID]
31      return $maxID
32  }
```

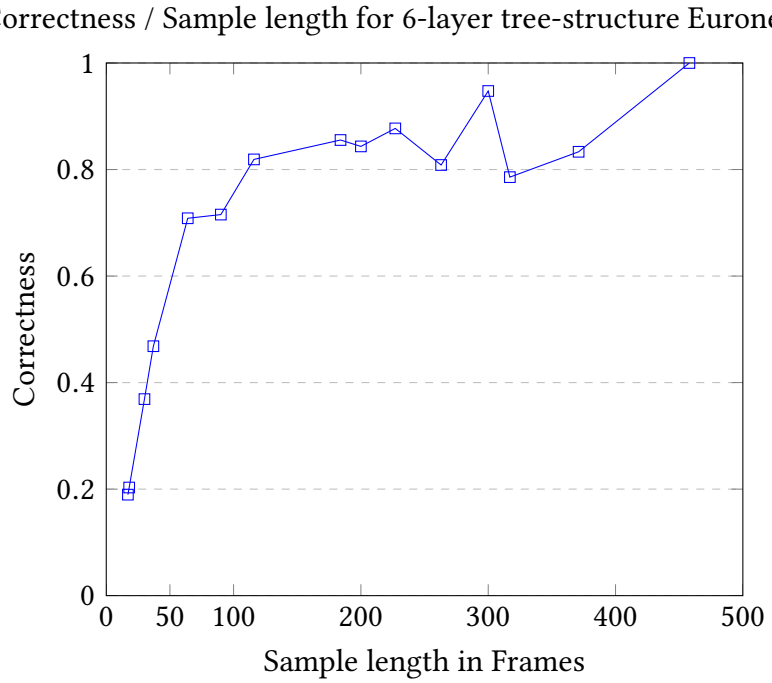
Listing 7.1: Most basic filter employed to smooth output

The results of the bare-net evaluation for Euronews can be seen in Table 7.1 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. 7.1 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.

As one can see a length of around 100 frames (stacked with the 33 frames context as described earlier), meaning a sample length of: $100 * 33/1000 \approx 3.3s$ 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 3.3 seconds (or around 5 seconds with a higher correctness of then about 80 %).

## 7.2. Basic Test Filter

The first filter tried was a 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 that. This of course means that the approach requires a 5 frame ($\approx 50ms$) "warm-up" time, which still would make it usable in an online environment. It did however provide no improvement over the bare network output.




| **Language** | **Error** |
|---|---|
| Arabian | 0.364 |
| German | 0.300 |
| Spanish | 0.320 |
| French | 0.300 |
| Italian | 0.326 |
| Polish | 0.267 |
| Portuguese | 0.291 |
| Russian | 0.315 |
| Turkish | 0.337 |
| English | 0.267 |
| **Overall** | 0.267 |

Table 7.1.: Results of the best net structure (tree-structure 6-layer) evaluated on the Euronews Development Set.

Table 7.2 shows the result of the basic filter. Overall the error 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.

| Language | Error |
|---|---|
| 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 |

Table 7.2.: Results of the best net structure (tree-structure 6-layer) evaluated on the Euronews Development Set with the Basic Filter.

## 7.3. 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          nnFILTER   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          nnFILTERREAL 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 correct id. Results are listed in Table 7.7. As for the basic test filter, the bigger advanced filter offered a tiny improvement for some languages, ( 0.1% for both Portuguese and English) but lost out in all other languages. Detailed per-Language results for this and the following filters can be found in App. **??**.

## 7.4. 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 the amount of "sureness" the net had when determining the language) was bigger than 0.1.

This however, did not lead to an improvement in the robustness of the classification on the development set even with this small threshold, but rather an immense decline. The reason possibly being that the output of the 2nd 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. 7.7

## 7.5. 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 to 50, however both filters showed no improvement, but kept very close to the direct output of the net results, meaning that likely this was an improvement for longer samples but decreased accuracy in smaller samples as the output was likely to jump around anyways. As per-language results did not improve The full code of this filter can be found in the Appendix.

## 7.6. Two-Language Setup

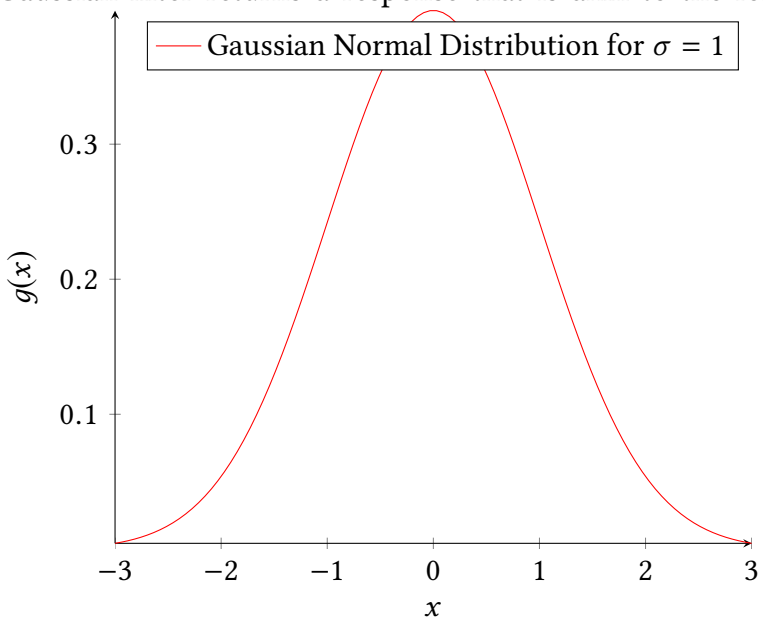
Table 7.6 shows the result produced by using the LID Euronews net for 10 languages on different combinations of 2 languages (namely French/Italian and English/German). In this case we ignore the output of the net if it doesn't equal one of the two languages and instead keep the previous output intact in this case. This, as intuition predicted, 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. 7.1.

| **Language** | **Error** |
|---|---|
| **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 |

Table 7.3.: Results of the best net structure (tree-structure 6-layer) evaluated on only 2 Languages from the Euronews Development Set.

## 7.7. Gaussian Smoothing Filter

Since most of the smoothing techniques tried out showed no significant improvement we decided to try a Gaussian filter as well. 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 discrete case $\sigma$, 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 $\sigma$, the Gaussian function for point $x$ can be calculated as:

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

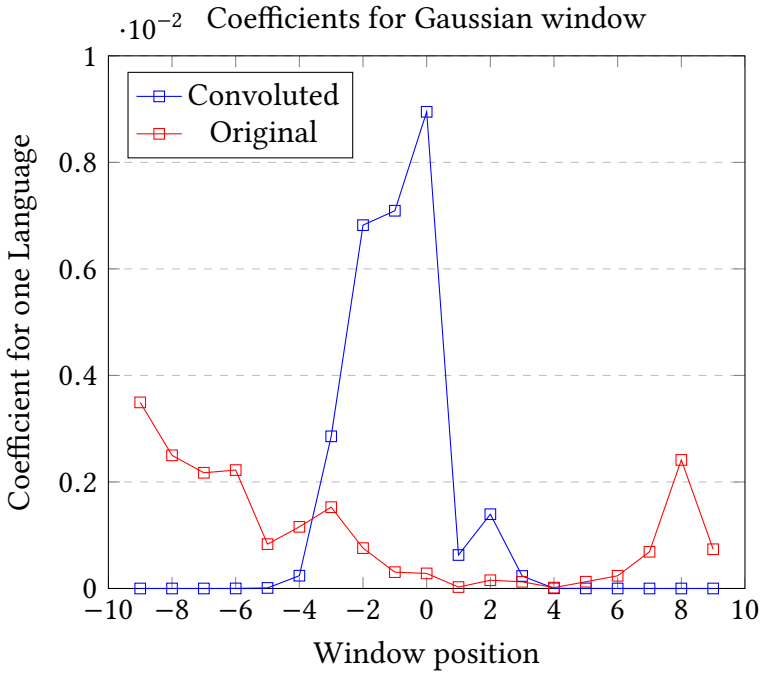
Once the Gaussian function is calculated for the window, the convolution of the data points with the Gaussian kernel is performed ,resulting in smoothed over coefficients of the LID net output. The maximum smoothed coefficient is then chosen and the corresponding language is output, which gave us definite improvements over not smoothing the net output at all. In tcl the relevant code part can be seen in 7.2, 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.

```
1    #Windowsize definition
2    set ws 5
3    set sigma [expr sqrt($ws/(2*[Pi]))]
```

```
4     #calculate gauss kernel
5     for {set k 0} {$k <= $ws} {incr k} {
6         set gauss($k) [expr (1/(sqrt(2*[Pi])*$sigma)) * ([E] ** - (( $k
              ** 2)/ (2 * $sigma ** 2)))]
7     }
8     #For each frame
9     for {set i 0} { $i < [featureSetLID frameN nnBNF]} {incr i} {
10        for {set j 0} {$j < 10} {incr j} {
11            set values($j) [lindex [featureSetLID frame nnBNF $i] $j]
12            set curValue 0
13            #go from $i - $ws to $i + $ws
14            for {set k [expr -$ws]} {$k < $ws} {incr k} {
15                #Only use the current value if it actually exists
16                if {[expr $i + $k] >= 0 && [expr $i + $k] < [
                      featureSetLID frameN nnBNF]} {
17                    set cur [lindex [featureSetLID frame nnBNF [expr $i
                          + $k]] $j]
18                    #get correct index for gauss kernel, as we only
                          calculated once above for symmetric function
19                    if {$k < 0} {
20                        set curIdx [expr -$k]
21                    } else {
22                        set curIdx $k
23                    }
24                    #running sum of all previous values in the window
                          convoluted with the gaussian kernel.
25                    set curValue [expr $curValue + $cur * [set gauss(
                          $curIdx)]]
26                }
27            }
28            #save the value for current language coefficient
29            set values($j) $curValue
30        }
31        #Find maximum on smoothed values
32        set max -1
33        set maxID -1
34        for {set j 0} {$j < 10} {incr j} {
35            if {[set values($j)] > $max} {
36                set max [set values($j)]
37                set maxID $j
38            }
39        }
40        #Set the output as the Language with Max. coefficient.
41        set currentOutput $maxID
42    }
```

Listing 7.2: Gaussian Smoothing Filter as implemented in tcl/tk for the jrtk

We tried different window sizes of which the result can be seen in Tab. 7.7. To show the correct implementation Fig. 7.7 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 occurred.




| Filter | Overall Error |
|---|---|
| Bare Net | 0.311 |
| Basic Filter | 0.318 |
| Advanced Filter Small | 0.313 |
| Advanced Filter Big | 0.316 |
| Difference | 0.436 |
| Gaussian Filter WS 5 | 0.313 |
| Gaussian Filter WS 10 | 0.313 |
| Gaussian Filter WS 15 | 0.313 |
| Gaussian Filter WS 20 | 0.313 |
| Counting Filter FR 10 | 0.316 |
| Counting Filter FR 50 | 0.316 |
| Counting Filter FR 100 | 0.313 |
| Counting Filter FR 150 | 0.313 |
| Speech/Noise Filter | 0.313 |
| English/German | 0.116 |
| French/Italian | 0.207 |

Table 7.4.: Results of the 6-layer tree-structure evaluated with all the tried post-filtering approaches.

## 7.8. Lecture Data

Of course, all the nets were tested against the lecture data corpus, as this is the closest to an actual possible integration into the KIT's lecture Translator. We added the same

filtering as for the 2-language setup, ignoring non-available languages which brought the recognition rate up to about 90 %. These results for different net setups are summarized in Tab. ??.

# 8. Conclusion

# 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.

[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.

[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 matter, complete source code listings as well as a Glossary at the end.

## A.1. Complete Source Code Listings

As mentioned in Ch. 7, we are listing the complete source code for all the different smoothing algorithms tried here.

```
proc filter {} {
    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($currentOutput)
    }
    }
    set maxOverall -1
    set maxID -1
    for {set i 0} {$i < 10} {incr i} {
    if {$total($i) > $maxOverall} {
         set maxOverall $total($i)
         set maxID $i
    }
    }
    puts -nonewline "Overall we have classified as: "
```

```
33      puts [getName $maxID]
34  }
```

Listing A.1: Basic (first try) Filter employed to smooth/improve output

```
1  proc filter {} {
2      for {set i 0} {$i < 10} {incr i} {
3      set totalM($i) 0
4      set count($i) 0
5      }
6      set lastFrameID -1
7      set counter 0
8      set currentOutput -1
9      for {set i 0} { $i < [featureSetLID frameN nnBNF]} {incr i} {
10     set maxFrame [lindex [lsort -decreasing -real [featureSetLID frame
           nnBNF $i]] 0]
11     set maxFrameID [lsearch -real [featureSetLID frame nnBNF $i]
           $maxFrame]
12     for {set j 0} {$j < 10 && $i < [featureSetLID frameN nnBNF]} {incr j
           } {
13         set maxFrame [lindex [lsort -decreasing -real [featureSetLID
               frame nnBNF $i]] 0]
14         set maxFrameID [lsearch -real [featureSetLID frame nnBNF $i]
               $maxFrame]
15         set count($maxFrameID) [expr {[set count($maxFrameID)] + 1}]
16         incr i
17     }
18     set maxAvg -1
19     set maxID -1
20     for {set k 0} {$k < 10} {incr k} {
21         if {[set count($k)] > $maxAvg} {
22             set maxAvg [set count($k)]
23             set maxID $k
24         }
25     }
26     set currentOutput $maxID
27     if {$currentOutput != -1} {
28                 set totalM($currentOutput) [expr {[set totalM(
                       $currentOutput)] + 1}]
29     }
30     }
31     set maxOverall -1
32     set maxID -1
33     for {set i 0} {$i < 10} {incr i} {
34     if {[set totalM($i)] > $maxOverall} {
35         set maxOverall [set totalM($i)]
36         set maxID $i
37     }
38     }
39     puts -nonewline "Overall we have classified as: "
40     puts [getName $maxID]
41     return $maxID
42 }
```

Listing A.2: Counting Filter employed to smooth/improve output

```
proc filter {} {
    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 - $max2Frame >= 0.1}]} {
       incr counter
       if {$counter >= 5} {
          set currentOutput $maxFrameID
       }
    }
    if {$currentOutput != -1} {
            set totalM($currentOutput) [expr {[set totalM(
               $currentOutput)] + 1}]
    }
    }
    set maxOverall -1
    set maxID -1
    for {set i 0} {$i < 10} {incr i} {
    if {[set totalM($i)] > $maxOverall} {
        set maxOverall [set totalM($i)]
        set maxID $i
    }
    }
    puts -nonewline "Overall we have classified as: "
    puts [getName $maxID]
    return $maxID
}
```

Listing A.3: Difference Filter employed to smooth/improve output

```
proc filter {} {
    for {set i 0} {$i < 10} {incr i} {
        set totalM($i) 0
    }
    set lastFrameID -1
```

```
6       set counter 0
7       set currentOutput -1
8       for {set i 0} { $i < [featureSetLID frameN nnBNF]} {incr i} {
9           set maxFrame [lindex [lsort -decreasing -real [featureSetLID
                frame nnBNF $i]] 0]
10          set maxFrameID [lsearch -real [featureSetLID frame nnBNF $i]
                $maxFrame]
11          set currentOutput $maxFrameID
12          if {$currentOutput != -1 && [featureSetLID frame SPEECH $i] == "
                1.000000e+00"} {
13              set totalM($currentOutput) [expr {[set totalM($currentOutput
                    )] + 1}]
14              set lastFrameID $currentOutput
15          } elseif {$lastFrameID != -1} {
16              set totalM($lastFrameID) [expr {[set totalM($lastFrameID)] +
                    1}]
17          }
18      }
19
20      set maxOverall -1
21      set maxID -1
22      for {set i 0} {$i < 10} {incr i} {
23          if {[set totalM($i)] > $maxOverall} {
24              set maxOverall [set totalM($i)]
25              set maxID $i
26          }
27      }
28      puts -nonewline "Overall we have classified as: "
29      puts [getName $maxID]
30      puts -nonewline "Length of sample: "
31      puts [featureSetLID frameN nnBNF]
32      return $maxID
33  }
```

Listing A.4: Speech Filter employed to smooth/improve output