TDK dolgozat

Fodor Ádám
Kopácsi László

Eötvös Loránd University

Faculty of Informatics

Department of Software
Technology And Methodology

Speech de-identification with deep neural networks

Conference of Scientific Students' Association

Supervisors:

Dr. habil. András Lőrincz
Senior Researcher
Department of Software
Technology And Methodology

Zoltán Ádám Milacski
Teaching Assistant
Department of Software
Technology And Methodology

Authors:

László Kopácsi
Computer Science Msc
second-year student

Ádám Fodor
Computer Science Msc
second-year student

Budapest, 2018.05.10.

Abstract

The privacy of the speaker is an important legal concern when cloud-based speech services are used. For this problem we suggest a deep recurrent neural network solution that removes personal characteristics from human speech by converting it to the voice of a Text-to-Speech system. The network transcodes between human and generic sequences of vocoder parameters, delta and delta-delta features. We empirically tested the method on multiple data sets including the TIMIT, the NTIMIT, the LibriSpeech and the CSLU Kids' Speech corpora. To get the best results we compared multiple TTS systems, vocoders and audio alignment techniques. We evaluated the sound quality with two state-of-the-art Automatic Speech Recognition systems: the Google Speech API and the Wav2Letter neural network. We measured the success of de-identification by conducting a human estimation experiment. This novel speech evaluation approach avoids the human assessment of the perceived quality of the results.

Contents

Chapter 1

Introduction

De-identification is the process to remove any personal information from the data, that can be associated with a person's identity. This can be a name if unique enough, national identification number, handwriting, medical records, face or even voice. Without de-identification using data can have legal concerns.

The cloud-based services are improving in an incredible pace, in many fields they already reached superhuman performance. In turn they usually store all the uploaded data, therefore their use can have legal aspects and they might compromise the user's privacy. Because of this, proper anonymization of the data before upload is crucial.

Voice anonymization can be achieved by several ways. By applying several sound distortion algorithms it can be anonymized but usually with enough samples they can be reversed. Using Automatic Speech Recognition (ASR) followed by a Text-to-Speech (TTS) system the samples can be fully anonymized, but this is a two step procedure and hence errors can accumulate between the steps. With voice conversion, the voice of a source speaker can be altered to sound like the target, but generally it has to be trained for each speaker.

We propose a method that can transform any speaker's voice to a target generic voice, using deep neural networks. By doing this the resulted transformation cannot be reversed, and the trained network can be applied anywhere the privacy of the user is matters. In this paper several architecture with various features had tried out and the results had evaluated with 2 state-of-the-art ASR to achieve the best quality. The overall system can be used as an extra layer of protection to prevent information leakage, such us identity deduction and/or the utilization of sensitive data without the user's acceptance.

Chapter 2

Theoretical Background

The following sections walk through the essential definitions and concepts of the thesis, that help understanding the motivations behind the applied methods and decisions made later on. The overview starts with the fundamentals of speech processing, and continues with applications and applied methods for aligning time series. Last but not least, machine learning and in particular deep learning concepts are reviewed in detail, in addition to the used model architectures.

2.1 Speech Processing

2.1.1 Fundamental Frequency (F0)

The fundamental frequency is the frequency which our vocal folds vibrate. We perceive it as pitch. Figure 2.1 shows an example of extracted F0 using a sample from TIMIT.

2.1.2 Frequency Spectrum

The frequency spectrum [30] is the discrete Fourier transform (DFT) of an audio signal.

Let x_n be an audio signal. Then

$$X_k = DFT(x) = \sum_{n=0}^{N-1} x_n e^{-i\frac{2\pi kn}{N}}$$

is called the frequency spectrum of the signal x of length N.

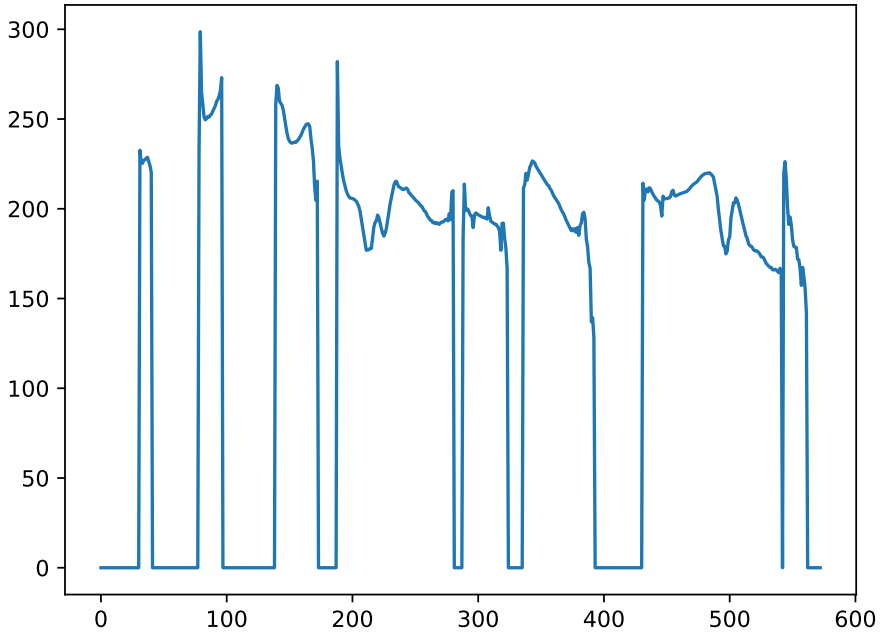

Figure 2.1: Fundamental frequency extracted from a TIMIT sample.

From this we can compute the *magnitude spectrum* M_k, the *phase spectrum* Φ_k and the *power spectrum* P_k:

$$\begin{aligned} M_k &= |X_k|\,, \\ \Phi_k &= \arctan\left|\frac{Re(X_k)}{Im(X_k)}\right|, \\ P_k &= Re(X_k)^2 + Im(X_k)^2. \end{aligned}$$

To convert the values of k into actual frequencies we can use the following formula:

$$f = \frac{k \cdot f_s}{N},$$

where f_s is the sampling frequency and N is the number of samples.

2.1.2.1 Spectral Envelope

The spectral envelope [30] is the contour of the magnitude spectrum. We can speak about upper and lower spectral envelope depending on which side it fits. The shape of this curve resembles the frequency response of the vocal tract. Formants and valleys could be easily read down. Figure 2.2 shows the spectral envelope of a violin.

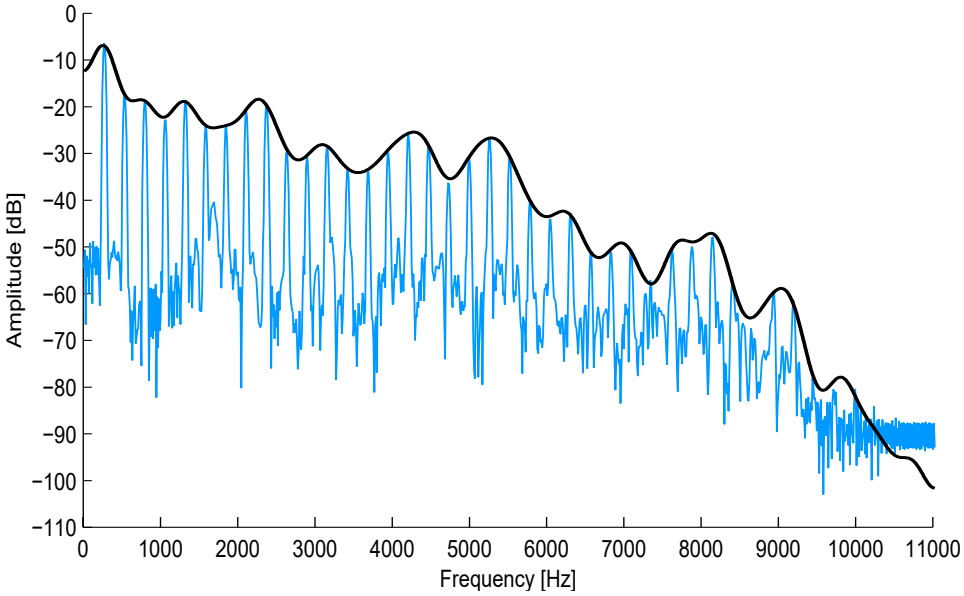

Figure 2.2: Spectral envelope of a violin [30].

2.1.2.2 Aperiodicity

The aperiodicity is the degree of randomness in a certain band of the magnitude spectrum. If we add it to the spectral envelope we get back the magnitude spectrum.

2.1.3 Cepstrum

The cepstrum is the inverse discrete Fourier transform (IDFT) of the logarithm of the audio signal's power spectrum:

$$C_n = IDFT(\log(P_k)).$$

It gives us a more compact, low dimensional, de-correlated representation, which is necessary when we are using statistical models, but with deep learning we could use the spectrum instead.

2.1.4 Mel-Cepstral Analysis

With mel-cepstral analysis [11, 35] we can warp the frequency scale and compress the frequency coefficients. With the following formula we can calculate the Mth order mel-cepstral coefficients:

$$\log\left(X(e^{-i\omega})\right) = \sum_{m=0}^{M} \tilde{c}_m e^{-i\tilde{\omega}},$$

where $X\left(e^{-i\omega}\right)$ is the discrete Fourier transform of x_n,

$$\tilde{\omega} = \tan^{-1} \frac{(1-\alpha^2)\sin\omega}{(1+\alpha)\cos\omega - 2\alpha}.$$

is the phase response of an all-pass filter. It gives us the warped frequency scale. The $\alpha \in [-1, 1]$ is the all-pass constant, which gives the phase characteristic. With the right α value the mel-scale becomes a good approximation to the human auditory system.

2.2 Vocoders

Vocoders are speech analysis and synthesis tools. With their help we can encode the raw waveform to a more compressed representation that takes the human auditory system into consideration. This is why the resulted features are generally better to work with. Unfortunately this compression sometimes results in loss of quality, e.g., artifacts may be introduced in the synthesized speech. Many vocoders exist, e.g., Ahocoder [9], MagPhase [10], PulseModel [7], WORLD [24] and STRAIGHT [20]. Moreover autoencoders also can be used as neural vocoders, e.g., SampleRNN [23] is used in Char2Wav [33].

2.3 Text-to-Speech systems

Speech synthesis is the transformation of text-to-speech. This conversion is done by a Text-to-Speech (TTS) system by design, that is as close to real speech as possible according to the pronunciation norms of special language. The input element of a TTS engine can be plain or annotated text as well as symbolic linguistic representation, and produces generic speech signal as its output.

Speech synthesis can be done by several different methods. The most widely used ones are:

Waveform concatenation: Individual segments of speech signal are concatenated during this process. The samples are obtained directly from human utterances. Speech produced by waveform concatenation can be very natural sounding, if any two segments to be concatenated have similar pitch periods and spectral envelops that match at the endpoint. Otherwise, degrading quality can be perceived caused by spectral discontinuities.

Statistical Parametric Speech Synthesis (SPSS): In SPSS, one of the most important factor that limits the naturalness of the synthesized speech is the so-called acoustic model, which learns the relationship between linguistic and acoustic features: this is a complex and non-linear regression problem. For the past decade, Hidden Markov Models (HMMs) have dominated acoustic modelling. Recently, neural networks have been "rediscovered" as acoustic models. Deep feedforward neural networks (DNNs) can be viewed as replacement for the decision tree used in the HMM-based speech synthesis.

There are several freely available toolkits for both approaches. The process of assigning phonetic transcriptions to words is called grapheme-to-phoneme conversion. The phonetic transcriptions and prosody information together make up the symbolic linguistic representation. This preprocessing of texts and waveform concatenation can be done with Festival, as well as SVOX pico2wave. In case of SPSS, the information for speech signal generation is given by the parameters of acoustic model. Free toolkits are for example eSpeak, WaveNet [36] and Merlin [12].

2.4 Automatic Speech Recognition systems

In the previous chapter, text-to-speech transformation was outlined briefly. Automatic Speech Recognition (ASR) represents the opposite direction: spoken utterances can be translated to text. In most cases speech recognition is regarded as a front-end for Natural Language Processing (NLP) components, such as acoustic or language model. Well-known examples are the YouTube closed captioning or the Siri front end. Building end-to-end ASR systems have been a big research interest since 2014 [40], because they can replace most modules with a single model.

During our experiments we measured sound quality with two state-of-the-art automatic speech recognition systems: the Google Cloud Speech API and the remarkable Wav2Letter neural network of Facebook. Both has advantages and disadvantages, too.

The ASR system of Google is available as a cloud-based service. It is one of the best commercial tools considering performance and quality, but it requires paid subscription. The primary asset of Google Speech API is that accuracy improves over time as Google improves the internal speech recognition technology.

Another state-of-the-art option is an end-to-end ASR system called Wav2Letter [6, 37] from Facebook AI Research. The acoustic model is a convolutional neural network with Gated Linear Units (GLUs). It was trained on LibriSpeech corpus and reached competitive results. Additionally the group provided pre-trained models and made the toolkit freely available.

Finally, there is a well-known speech recognition toolkit particularly designed for researchers: Kaldi [26]. The software is currently maintained and contains tried and

tested recipes from the most standard techniques of ASR systems. Montreal Forced Aligner in our experiments uses the Kaldi ASR toolkit as underlying technology.

2.5 Audio normalization

The overall volume of the audio can be changed to reach a target level with normalization. Accordingly, the loudness of the sound file can be maximized, or a group of audio files at different volumes can be set as close as possible to the same level.

Audio normalization techniques[1]:

Peak Volume Detection: Volume is set according to the largest peaks only.

Root mean square (RMS) Volume Detection: This algorithm considers the overall loudness of a file. Large peaks may be clipped.

EBU R-128 Volume Detection: Similar to the RMS method, it normalizes the perceived loudness of the audio file between 1000-6000 Hz.

The audio samples of the databases were recorded in controlled circumstances with negligible noise. Consequently, the Peak Volume Detection was used later on.

2.6 Forced Alignment

The algorithm aligns the audio file (containing speech data) with its transcript, identifying which time segments in the speech data correspond to the spoken text. Word- and phoneme-level alignment can be computed as a result (Figure 2.3). As a step of database preprocessing, the produced transcript alignments were visualized and validated with Praat (Figure 2.4).

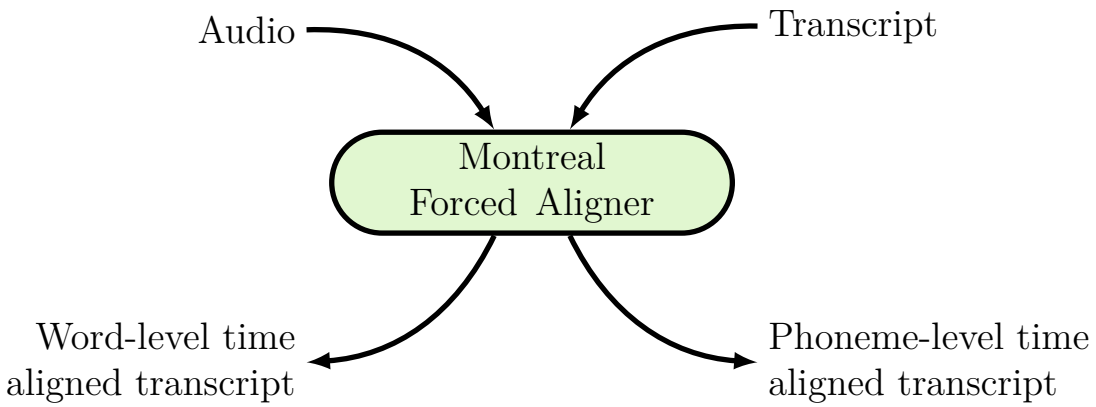

Figure 2.3: Montreal Forced Aligner.

[1] https://www.learndigitalaudio.com/normalize-audio

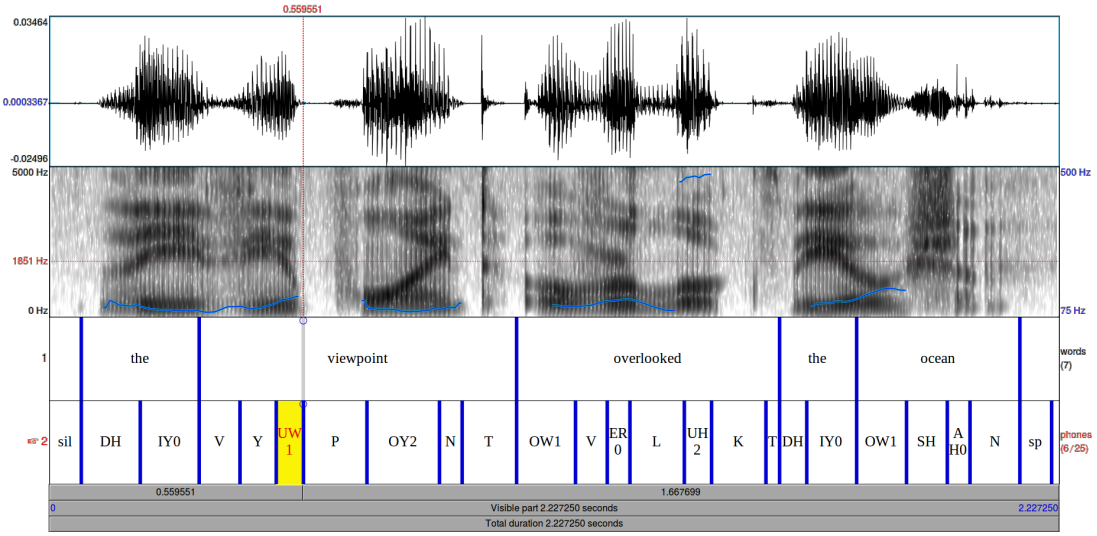

Figure 2.4: Visualizing word- and phone-level alignment with Praat. Transcript: "The viewpoint overlooked the ocean."

2.7 Dynamic Time Warping

As its name suggests dynamic time warping uses dynamic programming to find the optimal alignment between two time series (Figure 2.5).

More formally, if we have two sequences $X = (x_1, \ldots, x_N)$ and $Y = (y_1, \ldots, y_M)$, where $N, M \in \mathbb{N}$, and a similarity/distance measure $d(x, y)$, then we can determine the optimal warping path between them, denoted by $p^* = (p_1, \ldots, p_L)$ $(L \in \mathbb{N}, \forall l \in [1, L] : p_l = (n_l, m_l) \in [1, N] \times [1, M])$, as follows:

$$p^* = \underset{p}{\operatorname{argmin}} \sum_{l=1}^{L} d(x_{n_l}, y_{m_l}).$$

The trivial implementation has $O(N \cdot M)$ memory and time complexity, but there are approximate algorithms with $O(\min(N, M))$ complexity, like FastDTW [28].

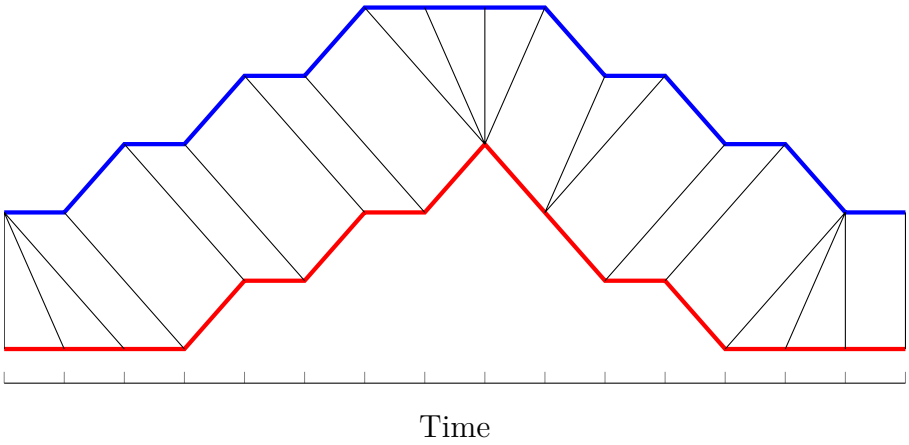

Figure 2.5: A possible alignment between two time series [28].

2.8 Deep Learning

Deep learning is a subfield of Machine learning, which uses deep neural networks (DNNs) to learn from data. Nowadays it can be found almost everywhere, from image classification to emotion recognition and of course speech processing. As the computing power is increasing and the datasets are getting bigger and bigger, these deep learning models are becoming much more accurate, and in many fields they reach superhuman performance as well.

Basically with DNNs we can learn an $\mathcal{X} \mapsto \mathcal{Y}$ transformation, where $\mathcal{X}$ is the set of input samples and $\mathcal{Y}$ is the set of output samples. The learning can be unsupervised, when $\mathcal{Y} = \mathcal{X}$, or supervised. In the latter case we can distinct classification, where $\mathcal{Y} = \{1, 2, \ldots, k\}^m$, and regression, where $\mathcal{Y} = \mathbb{R}^m$.

The learning is done by finding the $\boldsymbol{\theta}^*$ optimal parameter of the $f(\boldsymbol{\theta}, .)$ parametric transformation on a given $\left\{\left(\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}\right) \middle| \, n \in [1, N] \wedge \boldsymbol{x}^{(n)} \in \mathcal{X} \wedge \boldsymbol{y}^{(n)} \in \mathcal{Y}\right\}$ training set so that $f\left(\boldsymbol{\theta}^*, \boldsymbol{x}^{(n)}\right) \approx \boldsymbol{y}^{(n)}$ $(n = 1, \ldots, N)$. To achieve this, a loss function l is defined and by minimizing that, we can find this "best" $\boldsymbol{\theta}^*$ parameter for it:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{n=1}^{N} l\left(f\left(\boldsymbol{\theta}, \boldsymbol{x}^{(n)}\right), \boldsymbol{y}^{(n)}\right) + \lambda R(\boldsymbol{\theta}),$$

where $R(\boldsymbol{\theta})$ is the regularization term, which limits the parameter, therefore it can help to prevent overfitting, and $\lambda \geq 0$ is its weight.

$f(\boldsymbol{\theta}, .)$ may consist of multiple transformations, which are organized into layers:

$$\begin{aligned} \boldsymbol{h}_0^{(n)} &= \boldsymbol{x}^{(n)}, \\ \boldsymbol{h}_d^{(n)} &= f_d\left(\boldsymbol{\theta}_d, \boldsymbol{h}_{d-1}^{(n)}\right), \quad d = 1, \ldots, D, \end{aligned}$$

where D is the depth of the network and $\boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_D\}$ is its parameters.

There are many layer types we can choose from, e.g., fully-connected (Section 2.8.1), recurrent (Section 2.8.2) and convolutional (Section 2.8.3) layers. But to find the best architecture for a given problem is the art of deep learning.

Every layer has at least one activation function g_d. These are non-linearities, which play an important role in the training process, because they facilitate universal function approximation. There are multiple popular activation functions:

$$\begin{aligned} \mathit{linear}(x) &= x, \\ \sigma(x) = \mathit{sigmoid}(x) &= \frac{1}{1 + \exp^{-x}}, \\ \tanh(x) &= \frac{1 + \exp^{-2x}}{1 - \exp^{-2x}}, \\ \mathit{ReLU}(x) &= \max(0, x). \end{aligned}$$

With the loss function $l(\hat{\boldsymbol{y}}, \boldsymbol{y})$ we can measure the performance of the network. For different tasks we use different losses. Notable loss functions include:

- Classification

  Cross-entropy: $CE(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_{i=1}^{m} \boldsymbol{y}_i \log \hat{\boldsymbol{y}}_i$,

- Regression

  Mean Squared Error: $MSE(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{m} \sum_{i=1}^{m} (\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i)^2$,

  Mean Absolute Error: $MAE(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{m} \sum_{i=1}^{m} |\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i|$.

Using such loss functions we can update the parameters of the model. We usually do it using some kind of gradient descend:

$$\boldsymbol{\theta_{l+1}} \leftarrow \boldsymbol{\theta_l} - \alpha \frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta_l}},$$

where α is the learning rate. This procedure has its flaws, e.g., it can get stuck in saddle points. Hence recently several improvemets were proposed in the literature, which use an adaptive learning rate [8, 41, 21]. We can also decouple the calculation of the derivatives and speed up the training with synthetic gradients [17].

During the training it is possible that the model will overfit. To prevent this, many techniques were developed:

Early stopping [27]: if the validation error does not improve for a specified number of epochs, then we stop the training.

Regularization: with ℓ_2 and ℓ_1 we can avoid large weights and achieve sparse representations.

Data augmentation: apply some kind of noise on the dataset to multiply its size, and to get a more robust representation.

Dropout [34]: randomly "turn off neurons" during training, the idea behind this is to learn bypassing the noise.

2.8.1 Fully-connected neural network

The fully-connected neural network is the simplest deep neural network. It consist of multiple layers of neurons, in which each one is connected to all neurons from the previous layer. Such building blocks are also called dense layers. Each connection has its own weight. This is a general purpose connection pattern, that does not make any assumptions about the features of the data. It can learn a functional relationship between the inputs and outputs of the system:

$$\boldsymbol{h}_d^{(n)} = g_d(\boldsymbol{W}_d \boldsymbol{h}_{d-1}^{(n)} + \boldsymbol{b}_d),$$

where $\boldsymbol{\theta}_d = \{\boldsymbol{W}_d, \boldsymbol{b}_d\}$.

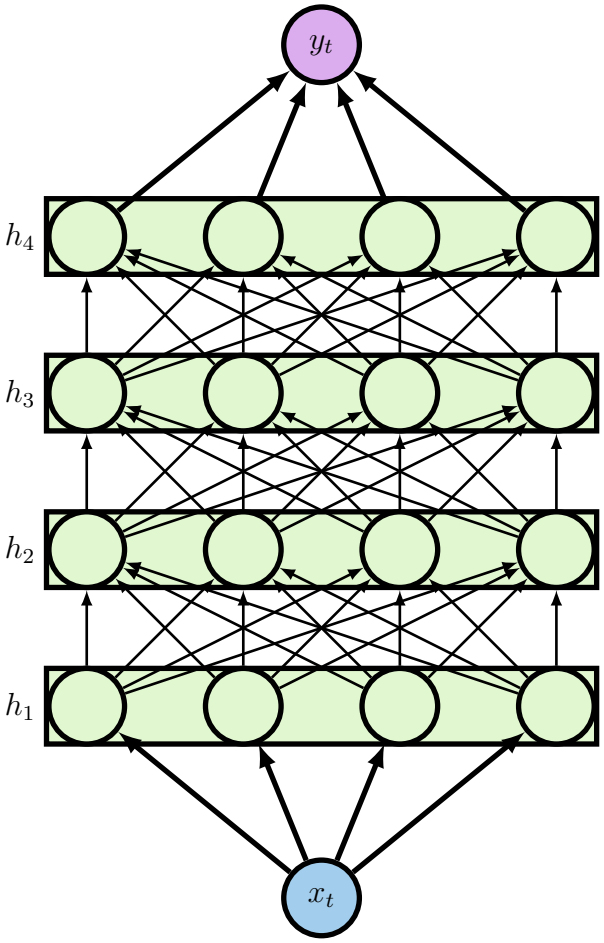

Figure 2.6: A feed forward neural network consisting of 4 dense layers.

2.8.2 Recurrent Neural Network

Recurrent neural networks (RNNs) became popular models over the years, that have achieved and shown great promise in many NLP tasks, such as text, handwriting and speech processing. RNNs have an internal hidden state, which is updated every time, when a new input is read by the system. Also the internal hidden state is fed

back to the model the next time it reads an input, and it is used as a past contextual information. RNNs are suitable for learning long-term dependencies, therefore they can be applied to time series and sequences.

A recurrent layer can be defined as:

$$\boldsymbol{h}_{d,t}^{(n)} = g_d\left(\boldsymbol{W}_d^h \boldsymbol{h}_{d,t-1}^{(n)} + \boldsymbol{W}_d^x \boldsymbol{h}_{d-1,t}^{(n)} + \boldsymbol{b}_d\right), \quad t = 1, \ldots, T_n,$$

where T_n is the length of the sequence, $\boldsymbol{h}_{d,t-1}^{(n)}$ is the state of the previous layer, and $\boldsymbol{\theta}_d = \left\{\boldsymbol{W}_d^h, \boldsymbol{W}_d^x, \boldsymbol{b}_d\right\}$ is a set of parameters of the layer.

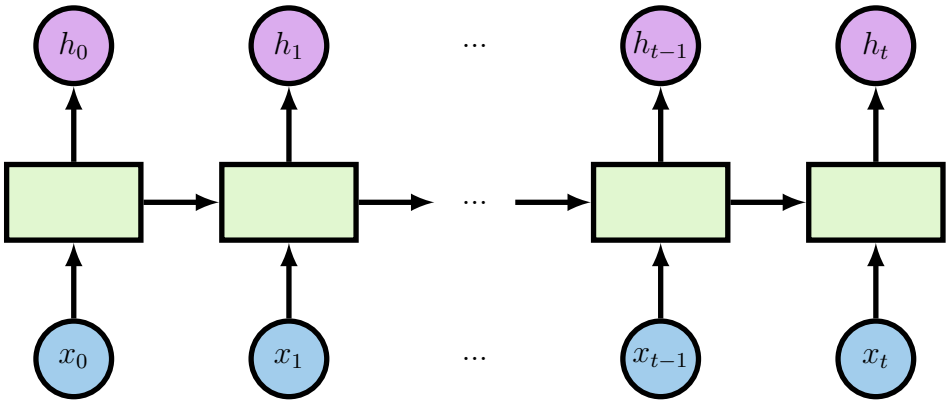

Figure 2.7: Recurrent neural network.

Training an RNN is similar to training a traditional neural network, backpropagation algorithm is used. However, the hidden states are shared, so the calculation of gradient in a current time step depends on the previous ones. So to calculate the gradient in $t = 4$, we have to backpropagate 3 steps to $t = 1$ and sum up the gradients. This is called Backpropagation Through Time (BPTT) [38].

BPTT can be computationally expensive as the number of time steps increases, furthermore the range of context is limited due to the vanishing gradient problem. There are many proposed solutions to solve this problem: the most widely used techniques are the Long Short-Term Memory (LSTM) [15] architecture, and its simplified version, the Gated Recurrent Unit (GRU) [4].

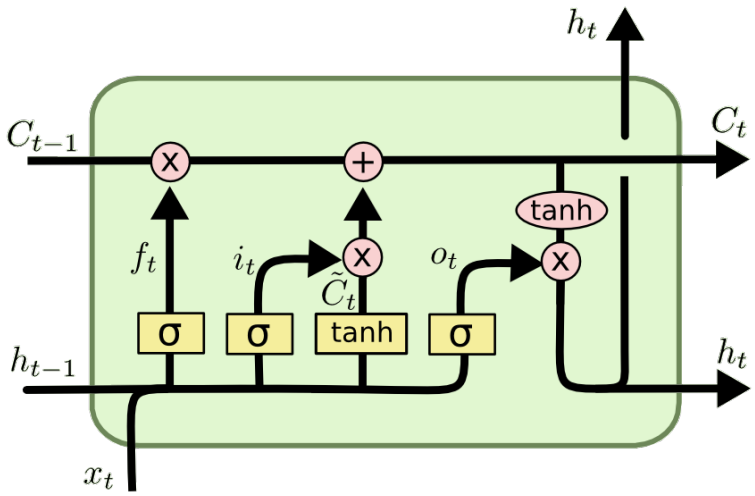

Figure 2.8: Long Short-Term Memory (LSTM)[2].

2.8.2.1 Long Short-Term Memory networks

Long Short-Term Memory [15] is a special kind of RNN, capable of learning long-term dependencies. An RNN is a chain of repeated modules. While such modules can be as simple as a single *tanh* activation function, LSTM building blocks have a more complex structure. The modules were designed to solve the problem of vanishing gradients through a gating mechanism.

The formal definition of the LSTM module is the following:

$$
\begin{aligned}
\boldsymbol{f}_{d,t}^{(n)} &= \sigma\left(\boldsymbol{W}_d^{hf}\boldsymbol{h}_{d,t-1}^{(n)} + \boldsymbol{W}_d^{xf}\boldsymbol{h}_{d-1,t}^{(n)} + \boldsymbol{b}_d^{f}\right), \\
\boldsymbol{i}_{d,t}^{(n)} &= \sigma\left(\boldsymbol{W}_d^{hi}\boldsymbol{h}_{d,t-1}^{(n)} + \boldsymbol{W}_d^{xi}\boldsymbol{h}_{d-1,t}^{(n)} + \boldsymbol{b}_d^{i}\right), \\
\boldsymbol{C}_{d,t}^{(n)} &= \boldsymbol{f}_{d,t} \circ \boldsymbol{C}_{d,t-1} + \boldsymbol{i}_{d,t} \circ \tanh\left(\boldsymbol{W}_d^{hc}\boldsymbol{h}_{d,t-1}^{(n)} + \boldsymbol{W}_d^{xc}\boldsymbol{h}_{d-1,t}^{(n)} + \boldsymbol{b}_d^{c}\right), \\
\boldsymbol{o}_{d,t}^{(n)} &= \sigma\left(\boldsymbol{W}_d^{ho}\boldsymbol{h}_{d,t-1}^{(n)} + \boldsymbol{W}_d^{xo}\boldsymbol{h}_{d-1,t}^{(n)} + \boldsymbol{b}_d^{o}\right), \\
\boldsymbol{h}_{d,t}^{(n)} &= \boldsymbol{o}_{d,t}^{(n)} \circ \tanh\left(\boldsymbol{C}_{d,t}^{(n)}\right), \qquad t = 1, \ldots, T_n,
\end{aligned}
$$

where $\boldsymbol{\theta}_d = \left\{\boldsymbol{W}_d^{hf}, \boldsymbol{W}_d^{xf}, \boldsymbol{b}_d^{f}, \boldsymbol{W}_d^{hi}, \boldsymbol{W}_d^{xi}, \boldsymbol{b}_d^{i}, \boldsymbol{W}_d^{hc}, \boldsymbol{W}_d^{xc}, \boldsymbol{b}_d^{c}, \boldsymbol{W}_d^{ho}, \boldsymbol{W}_d^{xo}, \boldsymbol{b}_d^{o}\right\}$, and $\circ$ is the element-wise product operator.

[2] `http://colah.github.io/posts/2015-08-Understanding-LSTMs`

2.8.2.2 Bidirectional Recurrent Neural Network

The idea behind Bidirectional RNN [29] is using two independent recurrent models together. The input sequence is fed in the proper time order for one network, and in reverse time order for another. The outputs of the networks are combined at each time step: usually concatenation is preferred, but summation is also a good alternative.

Bidirectional Recurrent Neural Networks have already proven their effectiveness in the domain of speech recognition. There is evidence that the context of the whole utterance is used to interpret what is being said rather than a linear interpretation. Bidirectional recurrent networks can be defined as:

$$\begin{aligned}
\overrightarrow{\boldsymbol{h}_{d,t}^{(n)}} &= \overrightarrow{g_d}\left(\overrightarrow{\boldsymbol{W}_d^h}\overrightarrow{\boldsymbol{h}_{d,t-1}^{(n)}} + \overrightarrow{\boldsymbol{W}_d^x}\overrightarrow{\boldsymbol{h}_{d-1,t}^{(n)}} + \overrightarrow{\boldsymbol{b}_d}\right) \\
\overleftarrow{\boldsymbol{h}_{d,t}^{(n)}} &= \overleftarrow{g_d}\left(\overleftarrow{\boldsymbol{W}_d^h}\overleftarrow{\boldsymbol{h}_{d,t+1}^{(n)}} + \overleftarrow{\boldsymbol{W}_d^x}\overleftarrow{\boldsymbol{h}_{d-1,t}^{(n)}} + \overleftarrow{\boldsymbol{b}_d}\right) \\
\boldsymbol{h}_{d,t}^{(n)} &= g_d\left(\overrightarrow{\boldsymbol{W}_d}\overrightarrow{\boldsymbol{h}_{d,t}^{(n)}} + \overleftarrow{\boldsymbol{W}_d}\overleftarrow{\boldsymbol{h}_{d,t}^{(n)}} + \boldsymbol{b}_d\right), \qquad t = 1, \ldots, T_n,
\end{aligned}$$

where "$\rightarrow$" means normal time order, while "$\leftarrow$" is associated with reverse time order. $\boldsymbol{\theta}_d = \left\{\overrightarrow{\boldsymbol{W}_d^h}, \overrightarrow{\boldsymbol{W}_d^x}, \overrightarrow{\boldsymbol{b}_d}, \overleftarrow{\boldsymbol{W}_d^h}, \overleftarrow{\boldsymbol{W}_d^x}, \overleftarrow{\boldsymbol{b}_d}, \overrightarrow{\boldsymbol{W}_d}, \overleftarrow{\boldsymbol{W}_d}, \boldsymbol{b}_d\right\}$ is a set of parameters of the layer.

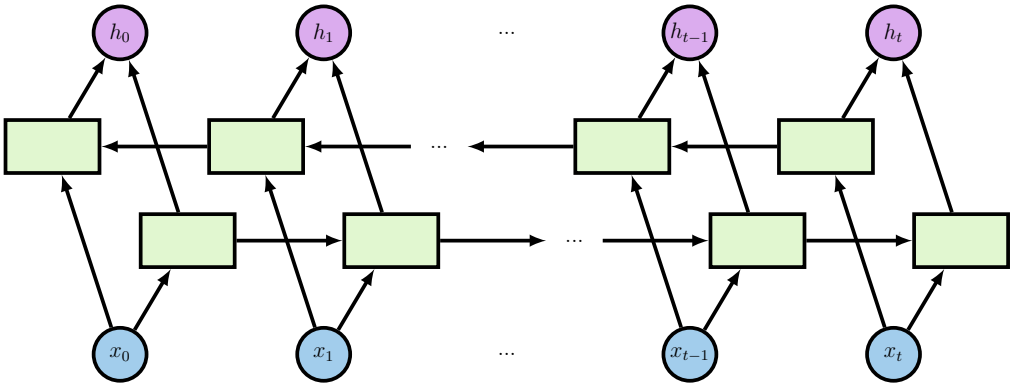

Figure 2.9: Bidirectional Recurrent Neural Network.

2.8.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs or ConvNets) are very similar to ordinary neural networks, however in a convolutional layer each neuron is only connected to a few nearby neurons in the previous layer. The advantage of this kind of architecture is the sharing of parameters. In this manner there are fewer parameters to estimate. This local connection layout with shared weights serve as filters.

Due to local connectivity, a CNN is able to learn local higher-level features from temporal or spatial data. ConvNets have achieved several breakthroughs in image

recognition tasks, while the idea is also applied in automatic speech recognition and utterance-level sentiment analysis in order to extract more representative acoustic features. Widely used input features of neural acoustic models in case of automatic speech recognition systems are MFCCs, power-spectrum, and mel-spectogram. In recent works, raw waveforms are utilized as input features without any preprocessing.

In speech processing, standard 1D convolutions can be applied to extract higher level representations of acoustic features. Let $(x_t)_{t=1..T_x}$ be an input sequence with T_x frames of d_x dimensional vectors. A convolution with kernel width kw, stride dw and d_y frame size output computes the following:

$$y_t^i = b_i + \sum_{j=1}^{d_x} \sum_{k=1}^{kw} w_{i,j,k} x^j_{dw \times (t-1)+k} \qquad \forall 1 \leq i \leq d_y,$$

where $b \in \mathbb{R}^{d_y}$ and $w \in \mathbb{R}^{d_y \times d_x \times kw}$ are the parameters of the convolution.

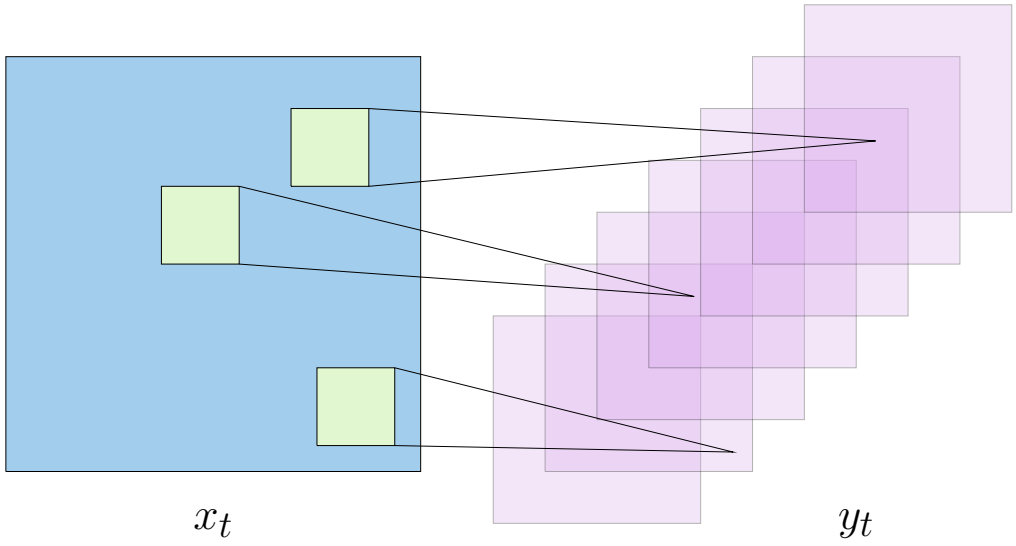

Figure 2.10: Convolutional Neural Network.

2.9 Hyper-Parameter Optimization

The performance of a model highly depends on its parameter settings. In case of Artificial Neural Networks, determining the architecture and the parameters of the learning algorithm can be difficult, because there are many various options, moreover the optimal setup might differ from problem to problem as well. Many parameters of the applied model need to be learned from data, but there are also higher level parameters, that can't be directly set during the regular training process. Hyper-Parameter Optimization (HPO) is the problem of optimizing a loss function over a graph-structured configuration space [3].
Some of the most popular methods are the following:

Grid search: The simplest algorithm that can be used for the optimization. After defining a set of parameter values, train the model for all possible combinations. The best model is returned as a result. It uses discrete values, so continuous variables need to be discretized first. The combinations of possibilities define a grid over the parameter space. The method can only work, if the model trains very quickly and there are only a few parameters, because of the curse of dimensionality.

Random search: In contrast to grid search, the trials are not on a grid. They are chosen randomly, which increases the overall effectiveness [2]. Random search does not cover the configuration space completely, therefore it is much quicker and does not rely on discretization. However, the method is not suitable for deep neural networks.

Hand-tuning: The idea is pretty straightforward. There are many options for trying out different configurations one after the other. E.g., after each training procedure, the performance of the model can be evaluated, and another set of parameter values can be chosen based upon the actual improvement. It is easy to see that hand-tuning is beneficial considering the number of trials, because humans can select parameters more effectively than grid or random search. So the question is the following: how can we automate the process?

Sequential Model-based Global Optimization: The success of the hand-tuning is based on the fact, that the previous results are analyzed before the next trial. Sequential Model-Based Global Optimization (SMBO) methods are used, when evaluation of the loss function is expensive. The surrogate function mimic most of the properties of the original objective-function, but it's cheaper to evaluate.

SMBO algorithms differ in the criteria by which it optimizes the surrogate, in addition to the way they model the surrogate. A widely used acquisition function is the Expected Improvement (EI), which is the expected probability that new trials will improve upon the current best observation[3].

Tree-structured Parzen Estimator (TPE) is a Sequential Model-based Global Optimization algorithm, which models $p(x|y)$ and $p(y)$, where y is the value of the surrogate function, x is the configuration. First of all, several observations are required before applying TPE algorithm. The initial configurations are generated with random search. Afterwards, the method collects new observations one by one and the algorithm decides which set of parameters it should try next. In every iteration, a distribution of the best observations are used to select the next candidate configuration. TPE uses parzen-window density estimators to model the distribution, and the tree-structured means that parameter space is defined in a form of a tree.

[3]https://arimo.com/data-science/2016/bayesian-optimization-hyperparameter-tuning/

Chapter 3

Methods

In this chapter, widely used measures are defined in the first place, then various components of the preprocessing pipeline are presented with detailed evaluations. After testing out different modules and techniques, decisions are made which ones will be used later on. Text-to-Speech systems are compared and as well as popular freely available vocoders. After selecting proper TTS and vocoder toolkits, time series alignments are reviewed with comprehensive evaluations and testing.

3.1 Measures

In speech recognition, the standard measure is the word error rate (WER), defined as the edit distance between the true word sequence and the most probable word sequence emitted by the transcriber.

Word error rate and word accuracy rate can be computed as:

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

$$WAR = 1 - WER = \frac{N - S - D - I}{N}$$

where, the N is the number of words in the reference, S is the number of substitutions, D is the number of deletions, I is the number of insertions.

However, it is not ideal for short sequences. In cases of 1-word sequences, where the transcriber recognizes the expected word, but mistakes 1-2 letters, the measure is 0%. This is not representative enough, that's why the Levenshtein distance (also called edit distance) is used instead.

Levenshtein distance is a measure of the similarity between two strings, which we will refer to as the source string (s) and the target string (t). The distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to transform s into t. This measure is also called character error rate (CER) or letter error rate (LER).

Consequently letter error rate is defined as:

$$LER = lev_{s,t}(i,j) = \begin{cases} \max(i,j), & \text{if } \min(i,j) = 0, \\ \min \begin{cases} lev_{s,t}(i-1,j)+1 \\ lev_{s,t}(i,j-1)+1 \\ lev_{s,t}(i-1,j-1)+1_{(s_i \neq t_j)} \end{cases} & \text{otherwise.} \end{cases}$$

where $1_{(s_i \neq t_j)}$ is the indicator function equal to 0 when $s_i = t_j$ and equal to 1 otherwise, and $lev_{s,t}(i,j)$ is the distance between the first i characters of s and the first j characters of t.

Letter accuracy rate (LAR) will be used in this document later on:

$$LAR = 1 - LER$$

3.2 Components of preprocessing

In this section, multiple available components are examined in contrast, so the best one is applied during database preprocessing. Determinant modules and methods are: the TTS system, vocoder toolkit and the applied audio aligner algorithm.

3.2.1 TTS comparison

The overall quality of the transformation is greatly depend on the TTS system we use as its target. For this reason we compared multiple popular, freely available software:

- the eSpeak with 12 Mbrola and 12 other voices,
- the Festival with 2 diphone and 11 clustergen US voices,
- the MaryTTS with 7 CMU (3 unit selection, 4 hidden semi-Markov model) and 8 DFKI (4 unit selection, 4 hidden semi-Markov model) voices and
- the SVOX pico with a British and an American voice.

The comparison pipeline can be seen on Figure 3.1. First we synthesized the specified text, then with the ASR we determined its transcript, finally we measured the mean accuracies. For the evaluation we used 192 TIMIT transcripts. A summary of the results can be found on Table 3.1. As we can see the Festival and the MaryTTS produced similarly good results with the same CMU RMS dictionary. Because of the ease of use, we chose the Festival.

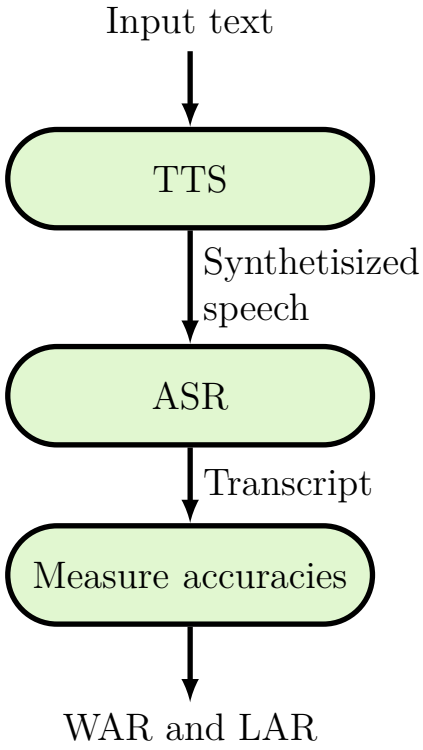

Figure 3.1: TTS comparison pipeline.

TTS name	**Voices (#)**	**Best WAR**	**Best LAR**
eSpeak	24	50%	74%
Festival	13	90%	97%
MaryTTS	15	90%	97%
SVOX pico	2	65%	85%

Table 3.1: Results of the TTS comparison.

3.2.2 Vocoder comparison

With the raw waveform is hard to work with. With vocoders we can get a better representation that takes into consideration the human hearing system. But this compression comes with a cost, eg. the loss of quality or the introductions of some artifacts in synthesized speech. To minimize its influence we examined several vocoder systems:

- Ahocoder [9]
  - logF0
  - cepstral coefficients
  - max voiced frequency

- MagPhase [10]
  - F0
  - magnitude spectrum
  - normalized real spectrum
  - normalized imaginary spectrum
- PulseModel [7]
  - interpolated F0
  - spectral envelope
  - phase distortion deviation (PDD)
  - noise mask: It's the thresholded PDD
- WORLD [24]
  - F0
  - spectral envelope
  - aperiodicity

We tested each vocoder on the TIMIT_test set. We took each sample, extracted vocoder parameters during the analysis step, then optionally compressed those with Mel-cepstral analysis, finally re-synthesized them, and measured their accuracies. This also can be seen on Figure 3.2.

Every method had less then 2% impact on the mean WAR and LAR values. As we can see this choice won't have a big impact on the system in terms of the output quality, we chose the one we prefer the most, the WORLD vocoder.

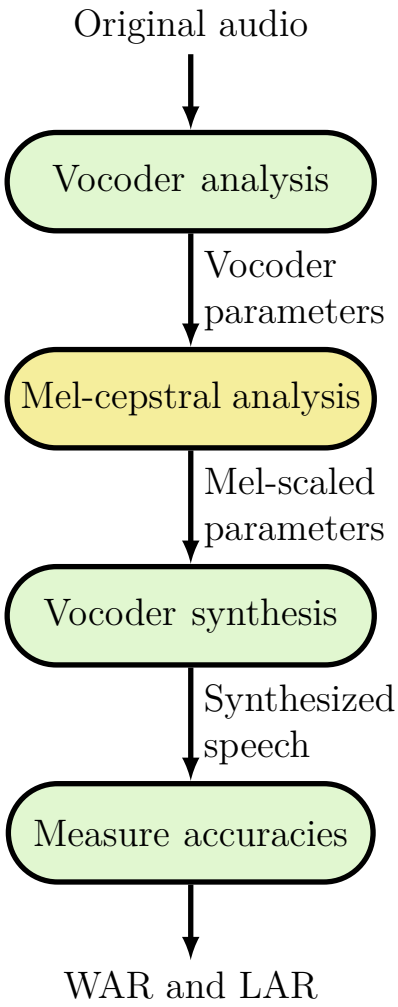

Figure 3.2: Vocoder comparison pipeline.

3.2.3 Alignment comparison

We aligned the original and text-to-speech audios to reduce the training difficulties introduced by the different sequence lengths. Two algorithms were applied: Dynamic Time Warping and Montreal Forced Alignment (MFA) with Waveform Similarity and Waveform Similarity Based Overlap-Add (WSOLA).

Applying DTW algorithm to raw waveforms might cause difficulties, therefore Mel-Generalized Cepstrum (MGC) were used instead. Euclidean distance were used to compute the cost matrix. The TTS voices of the test subset of TIMIT were transformed based on the optimal paths produced by DTW. In this manner the lengths of the original and the corresponding altered TTS sound files are equivalent.

Afterwards, the modified audio files were evaluated with Google Cloud Speech API, in order to get the predicted transcripts. Then the difference between the original and predicted transcripts were measured with word and letter accuracy rate, and presented in Table 3.2.

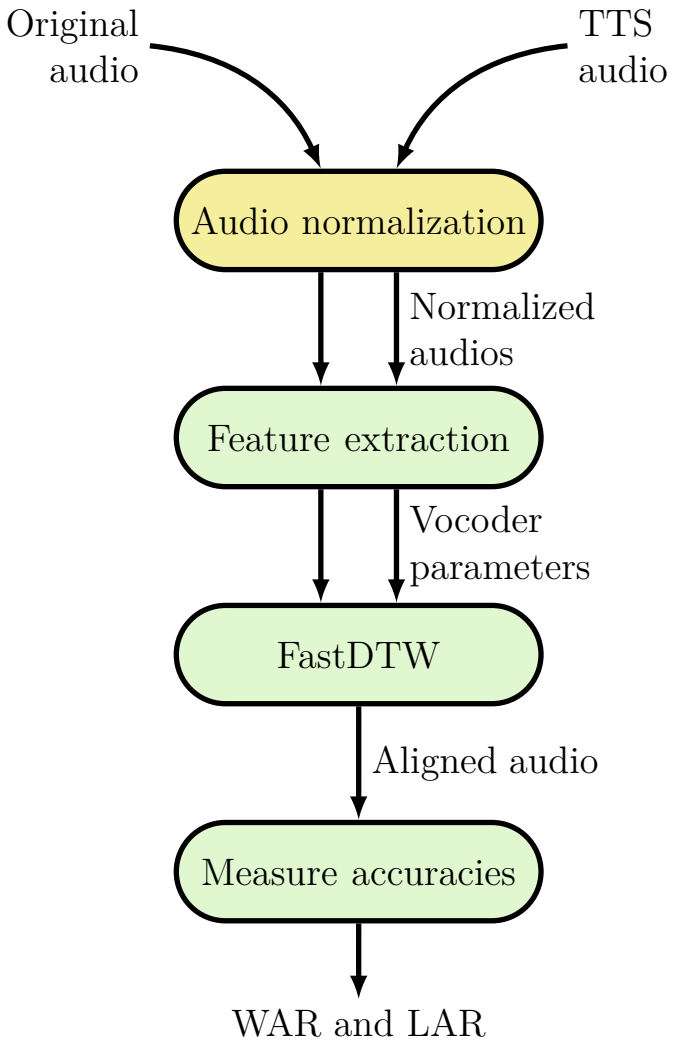

Figure 3.3: DTW alignment pipeline.

Name	**WAR GAPI**	**LAR GAPI**	**WAR W2L**	**LAR W2L**
TTS audio	90%	97%	83%	95%
DTW	81%	93%	79%	93%
DTW with peak normalization	78%	92%	76%	92%

Table 3.2: Performance of Google Cloud Speech API (GAPI) and Wav2Letter (W2L) after applying DTW algorithm on the TTS generated test audio samples of TIMIT corpus.

The method (Figure 3.3) produced fair results on the TIMIT database, but there were several misalignments in case of LibriSpeech. Audio normalization techniques (Section 2.5) were also applied before DTW, but these were not produced improvement.

The other approach is to determine the time intervals of the audio segments in case of every transcript, then use the time-aligned transcripts to align the audio samples. The length modification can be done with DTW or WSOLA algorithms.

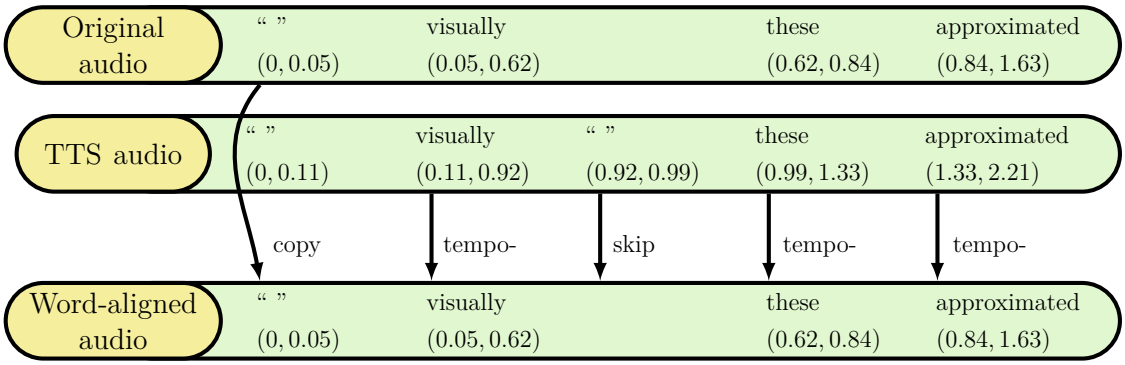

Figure 3.4: An example of creating word-aligned TTS audio sample.
Full transcript: “Visually, these approximated what he was feeling within himself.”

Figure 3.4 presents the process of creating word-aligned text-to-speech audio samples. First of all, Montreal Forced Aligner is used to pair intervals of speech segments with the corresponding transcripts in case of the original audio file, and the TTS sample as well. The goal is to properly synchronize the two file based on the intervals of words. The samples are sliced during the process, and combined or transformed with WSOLA (or DTW) to produce a new TTS sample with the same length as the original file.

We can differentiate several cases: the silence is copied from the original file, because we do not want to add dissimilar silence or generate noise. In case of matching words, the TTS segment is transformed with WSOLA algorithm to change the audio playback speed but not its pitch. On the figure, "tempo-" means a speed increase, because the TTS system produced the voice of the word a bit slower, than the speaker. Additional silences from the generic sound is not used.

Name	**WAR GAPI**	**LAR GAPI**	**WAR W2L**	**LAR W2L**
TTS audio	90%	97%	83%	95%
Phoneme-aligned audio with WSOLA	70%	88%	67%	87%
Word-aligned audio with WSOLA	78%	91%	76%	92%
Word-aligned audio with WSOLA then DTW	78%	92%	73%	91%

Table 3.3: Performance of Google Cloud Speech API and Wav2Letter after applying WSOLA algorithm on the TTS generated test audio samples of TIMIT database. Transformation took account of MFA intervals for transcripts and audio segments.

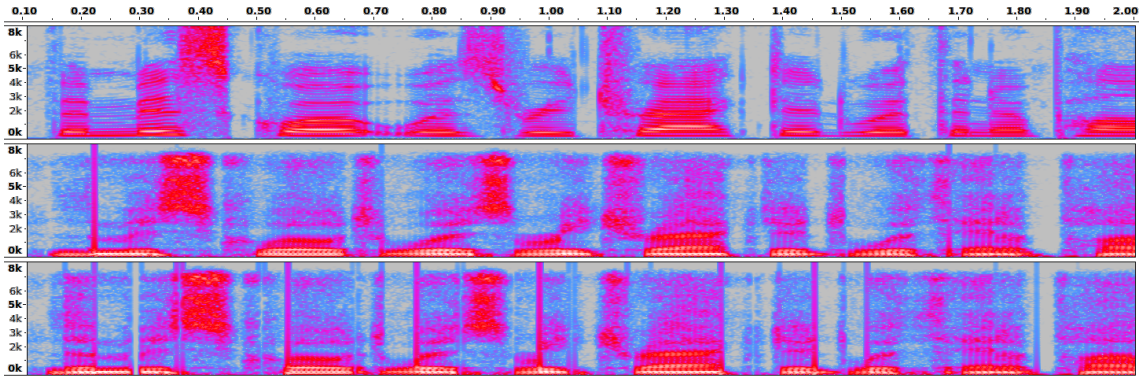

Figure 3.5: Spectograms of the original audio (top), word-aligned audio with WSOLA (middle) and phoneme-aligned audio with WSOLA (bottom)

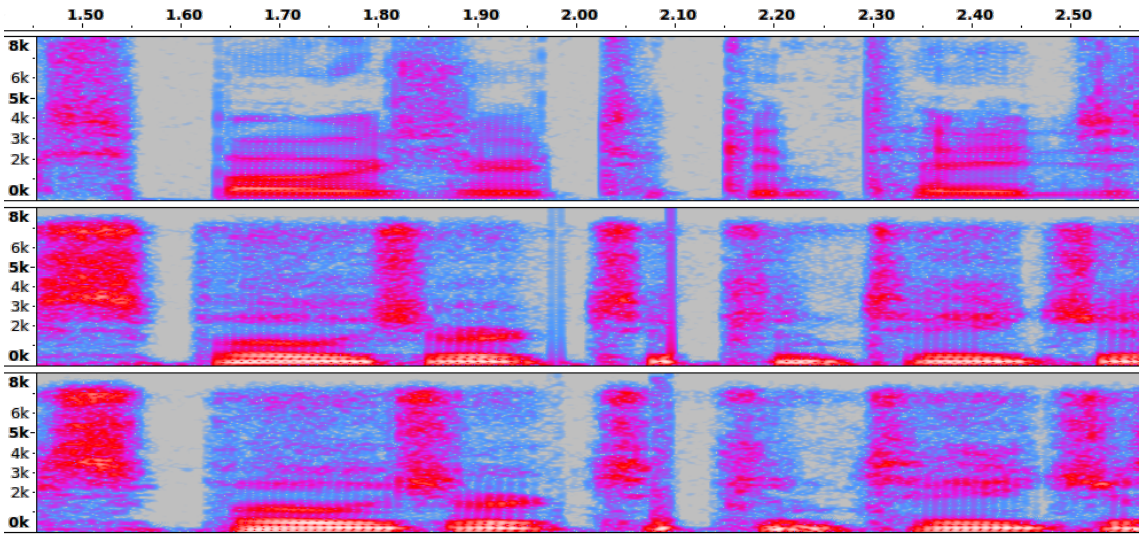

Figure 3.6: Spectograms of the original audio (top), word-aligned audio with WSOLA (middle) and word-aligned audio with WSOLA then DTW (bottom)

Afterwards, the modified segments are concatenated and evaluated with an ASR in order to get the accuracy measures. During slicing and concatenating several artifacts appeared in the transformed TTS sounds as crackles (Figure 3.5).

Table 3.3 shows, that the performance of ASR decreases on length-modified sound files especially in case of the phoneme-level alignment.

To remove the artifacts different techniques were tried as a post-processing step. One of the applied method was an extra DTW alignment. In this case, the TTS voice is modified with WSOLA algorithm using word-level alignment generated by Montreal Forced Aligner. Then execute a DTW alignment on the output of the WSOLA algorithm, to prevent the misalignments of a single DTW run and compensate the added noise of slicing and concatenations (Figure 3.6). The crackles were filtered out

successfully in most cases, but the accuracy measures only improved insignificantly (Table 3.3).

3.3 Datasets

The following paragraphs provide information about the benchmarking datasets.

3.3.1 TIMIT

TIMIT [42] is one of the most popular public corpus, which is used frequently for comparing different Machine Learning methods. The database is suitable for algorithm verification and parameter tuning due to its small size and phonetically diverse samples. It is designed for English phone recognition from continuous read speech.

The training set has 462 speakers, 8 utterances/speaker. The validation set consists of 50 speakers, totally 400 utterances, and the test set contains 192 sentences from 24 speakers. Each utterance is approximately 3.5 seconds long on average. The speakers also represent 8 major dialect regions of the United States.

3.3.2 NTIMIT

NTIMIT (Network TIMIT) [19] is a multi-speaker, telephone bandwidth speech database. The provided audio files contain continuous speech segments.

The NYNEX Science and Technology Speech Communication Group created the NTIMIT dataset using the original TIMIT samples. The audio files were collected by transmitting all 6300 TIMIT recordings through a telephone handset and over various channels in the NYNEX telephone network and redigitizing them: half of the recordings was sent over "local" telephone paths, while half was transmitted over "long distance" conditions. Transmission involved the use of a commercial device to simulate the acoustic characteristics between a human's mouth and a telephone handset.

The recording were done in an acoustically isolated controlled environment. Calibration signals were transmitted to each NYNEX central office in order to readily evaluate such network characteristics as attenuation, frequency response, and harmonic distortion.

Subset name	Samples (#)	Length (hours)
TIMIT_test	192	0.16
TIMIT_valid	1488	1.25
TIMIT_AUG_valid	4464	3.75
TIMIT_train	4620	3.94
TIMIT_train_half	2310	1.97
TIMIT_train_quarter	1155	0.98
TIMIT_AUG_train	13860	11.82
TIMIT_NTIMIT_test	384	0.32
TIMIT_NTIMIT_valid	2976	2.5
TIMIT_NTIMIT_train	9240	7.85

Table 3.4: Subsets of the TIMIT corpus [42].

3.3.3 LibriSpeech

LibriSpeech [25] is a 1000 hours long English corpus, derived from audiobooks. Each audio file is sampled at 16kHz. The database is freely available and widely used for comparing performance of speech recognition systems. Gender balance at the speaker level and in terms of amount of data available for each gender is ensured. The training set of the corpus is split into three subsets, with approximate size 100, 360 and 500 hours respectively. Distributing the partitions separately is more convenient for most users in this way.

The samples are more diverse regarding to the harmonics, than in case of TIMIT, NTIMIT and the CSLU Kids' Speech datasets. Furthermore, each audio file is longer than necessary, therefore they were split into several shorter parts to lighten the training of generated sequences.

Subset name	Length (hours)	Length/Speaker (minutes)	Female speakers	Male speaker	Total speakers
dev-clean	5.4	8	20	20	40
test-clean	5.4	8	20	20	40
dev-other	5.3	10	16	17	33
test-other	5.1	10	17	16	33
train-clean-100	100.6	25	125	126	251
train-clean-360	363.6	25	439	482	921
train-other-500	496.7	30	564	602	1166

Table 3.5: Subsets of the LibriSpeech corpus [25].

3.3.4 CSLU Kids' Speech Version 1.1

CSLU Kids' Speech Version 1.1 is a collection of spontaneous and prompted speech from 1178 children between Kindergarten and Grade 10 in the Forest Grove School District in Oregon. Each computer was manned by a CSLU staff member who monitored progress and helped the child with any difficulties. The average time at the computer was 20 minutes, yielding approximately 8-10 minutes of speech digitized at 16 bits and 16kHz using Soundblaster 16 PnP audio cards with head-mounted microphones.[31]

All children read approximately 60 items from a total list of 319 phonetically-balanced but simple words, sentences or digit strings. The total length of the database is 100 hours. The preprocessed parts of the scripted speech dataset is about 20 hours long.

A 2-hour-long subset was selected from the scripted collection considering the quality of the corresponding dynamic time warping transformed TTS sounds. The quality is measured with Google Speech API. The filtering was necessary, because most of the bad pairs of samples came from spoiled segments (created by cutting long audio files) or a wrong DTW transform. Using an ASR system can effectively filter the erroneous samples. However, filtering only the TTS sounds might be not enough: the original sound files of the children can be really noisy, quiet and mispronounced as well. In such cases there are no guarantee for producing correct anonymization, even if the target TTS voice is correctly made. The same "CSLU_KIDS_g_" prefix is used in the names of train, test and validation sets (Table 3.6) to sign that subset is heavily filtered considering the responses of Google Speech API.

Additionally, a 6-hour-long subset was generated from the quality-validated collection of audio files by applying speed perturbation augmentation technique. The method is explained in Section 3.5. Corresponding word-level transcriptions are also included.

Subset name	Samples (#)	Length (hours)
kids-1.1	73100	100.3
CSLU_KIDS_train	28245	14.04
CSLU_KIDS_test	6052	3.0
CSLU_KIDS_dev	6053	3.02
CSLU_KIDS_g_train	3307	1.73
CSLU_KIDS_g_test	300	0.15
CSLU_KIDS_g_dev	300	0.15
CSLU_KIDS_aug_train	9921	5.19
CSLU_KIDS_aug_test	900	0.45
CSLU_KIDS_aug_dev	900	0.45

Table 3.6: Subsets of the CSLU Kids' corpus [32].

3.4 Preprocessing

The pipeline of database preprocessing includes several steps described in the previous sections:

1. Converting audio files

   All audio files were converted to WAV format with sndfile-convert. This step filtered out a few unreadable files.

2. Preparing transcripts

   The files including the transcripts of TIMIT and NTIMIT recordings start with time intervals. However, they are irrelevant now, and being discarded.

   In case of LibriSpeech, there is only one text file for each speaker, which contains the corresponding transcripts of the audio files. One text file was generated for every audio, by processing it line by line.

   The creators of CSLU Kids' Speech dataset provided a set of (ID, transcript) pairs. The name of each audio file contains the identifiers of the speaker and the spoken transcript as well. One transcript file is generated for every recording considering the IDs.

3. Producing generic audio samples with Text-To-Speech system

   Festival Speech Synthesis System was used to generate TTS audio samples from transcript. The decision was made based on the results of TTS comparison in Section 3.2.1.

4. Aligning audio samples

   The TTS generated sound files were aligned to match with the corresponding sound file produced by the speaker. The alignment transformation was done by DTW using the mel-generalized cepstral coefficients.

5. Applying audio normalization

   In case of TIMIT, NTIMIT and LibriSpeech datasets, audio normalization is not necessary. However, most samples of the CSLU Kids' Speech corpus is inaudible, therefore peak normalization technique was applied based on the volume of TTS generated sounds.

6. Partitioning datasets

   Each dataset was split into train, validation and test sets. In the previous subsections, the used subsets of the files were presented.

7. Extracting vocoder features

   Figure 3.7 presents the pipeline of acoustic feature extraction. Fundamental frequency (F0), aperiodicity and spectral envelope is estimated with WORLD vocoder. Mel-generalized cepstrum and band aperiodicity were calculated from spectral envelope and aperiodicity respectively using mel-cepstral analysis. The logF0 is the logarithm of fundamental frequency. Linear interpolation of logF0 was calculated as well, with a binary vector called voiced/unvoiced (V/UV) mask (the value of the mask is 0, if F0 is zero in the examined frame, and 1 otherwise). Dynamic features are determined using mel-generalized cepstrum and band aperiodicity. All of the features signed with green color in the figure was extracted and used during training. Interpolated logF0, V/UV mask vector and dynamic features were tested as well.

8. Applying feature normalization

   Z-score normalization was applied to all of the calculated features, so they had the properties of a standard normal distribution with $\mu = 0$ and $\sigma = 1$.

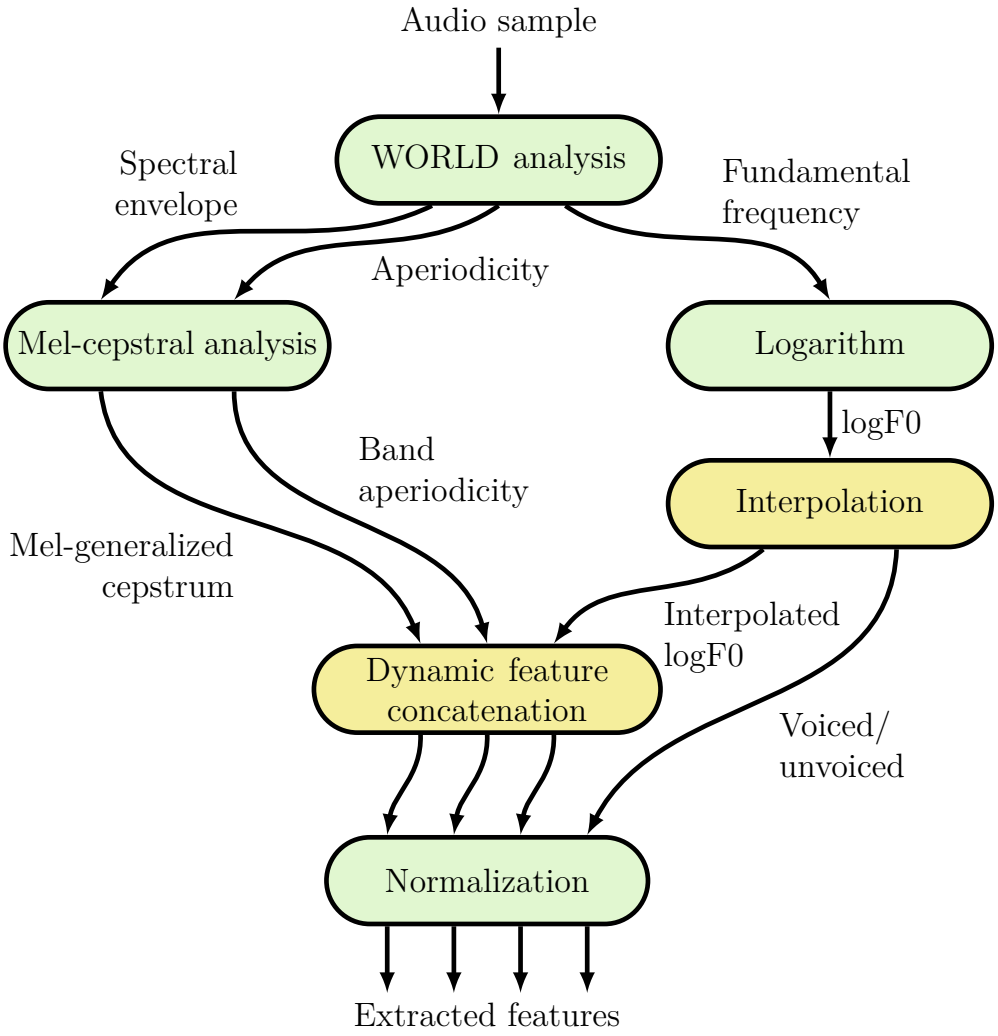

Figure 3.7: Feature extraction.

3.5 Audio augmentation

Data augmentation is a widely used technique in many machine learning tasks, such as image classification or speech processing, to increase the quantity of training data, and avoid overfitting. The enlargement can be accomplished with several different ways. In automatic speech recognition applying transformations and adding noise makes the model to learn a function that is robust enough to successfully handle several common effects [13].

Navdeep Jaitly and Geoffrey E. Hinton proposed an approach to augment training database using a random linear warping along the frequency dimension of spectrograms. A random warp factor for each utterance during training leads to significant gains. The technique is called Vocal Tract Length Perturbation (VTLP) [18].

Speed perturbation was recommended as well for audio data augmentation [22]. It is based on VTLP, however the authors of the article used WSOLA algorithm to

create two extra copy of each audio sample in case of using 0.9, 1.0, 1.1 warping factors. Data augmentation using various speed warping factors proved to be efficient, so we have applied it to generate larger TIMIT and CSLU Kids' Speech datasets.

3.6 Deep neural network

In Section 2.8, popular deep neural networks were presented, which provides different solutions to various problems. During the thesis the following deep architectures were used: Dense (Figure 3.8), which consists of four dense layers with 1024 units in each layer, and ReLU activation functions. These layers are signed as rectangles and followed by a dropout layer with 0.3 probability. The output is predicted after a dense layer with linear activation, which was used by the other models as well.

The BLSTM model contains two BLSTM layers with 384 units followed by 2×1024 dense layers (Figure 3.10). The used ConvNet has two 1D convolutional layers with 512 units and kernel width 7. Stride is set to 1. The filters are followed by 2 dense layers with the usual dropout and dense to produce the output (Figure 3.9).

A combined convolutional blstm model was tested as well (Figure 3.11). This model consist of 3×3 filters with 256 units and kernel width 3. On the output of every second filters a batch normalization was applied. After the batch normalizations [16] additional dropout were used with 0.25 probability. The convolutional layers were connected to dense layers with 1024 units.

A network based on ResNet18 [14] containing residual blocks (Figure 3.12) and a simplified Wav2Letter [6] architecture (Figure 3.13) were also used.

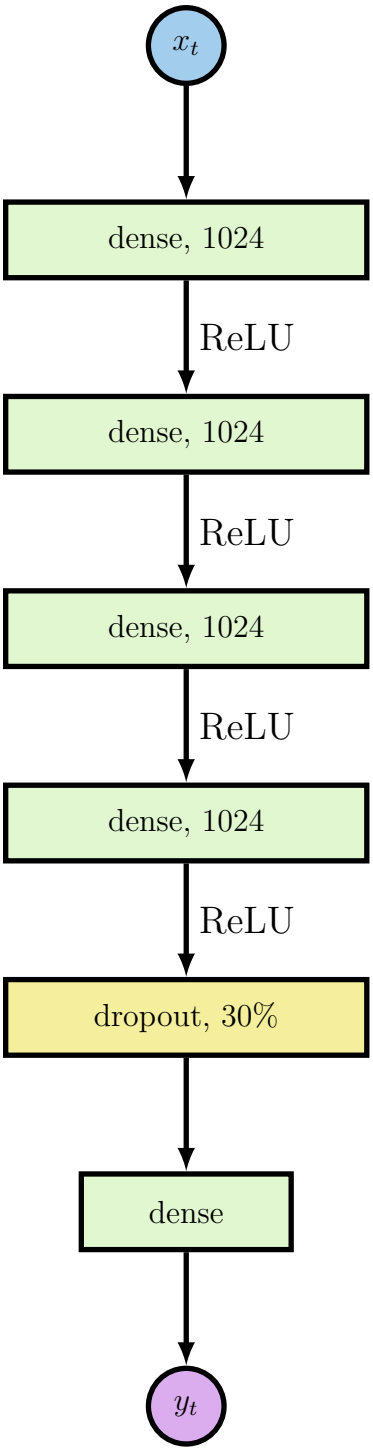

Figure 3.8: Dense architecture.

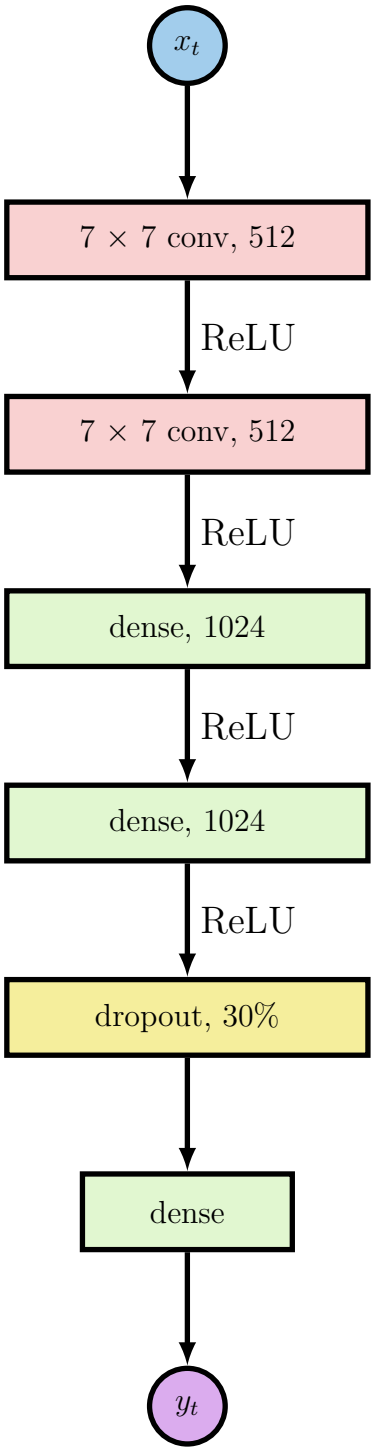

Figure 3.9: CNN architecture.

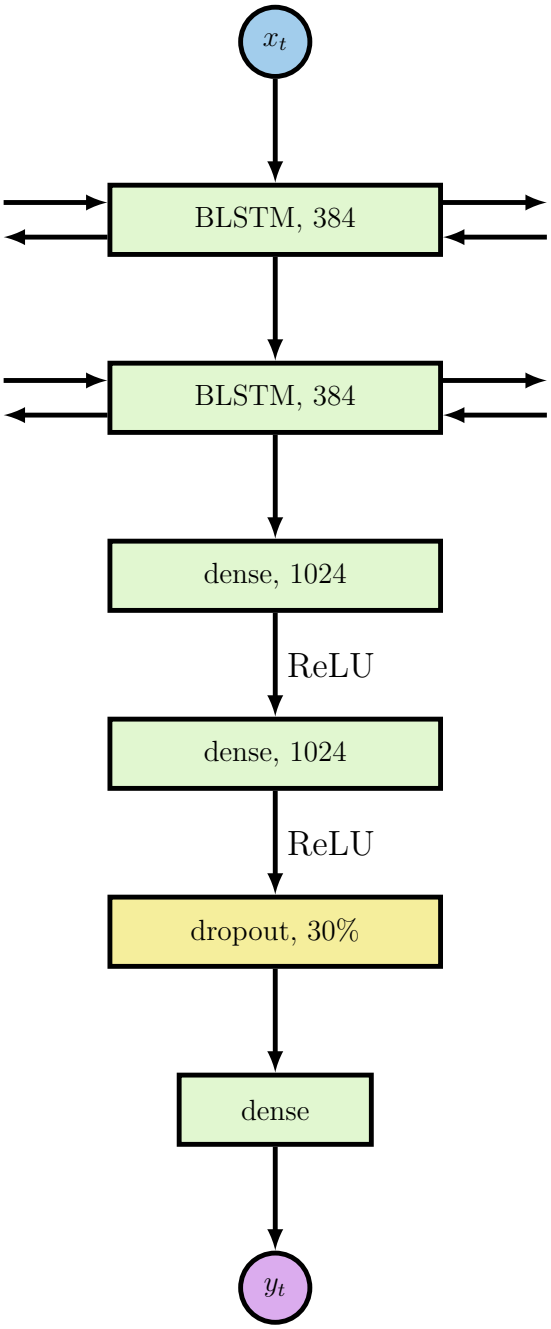

Figure 3.10: BLSTM architecture based on [12].

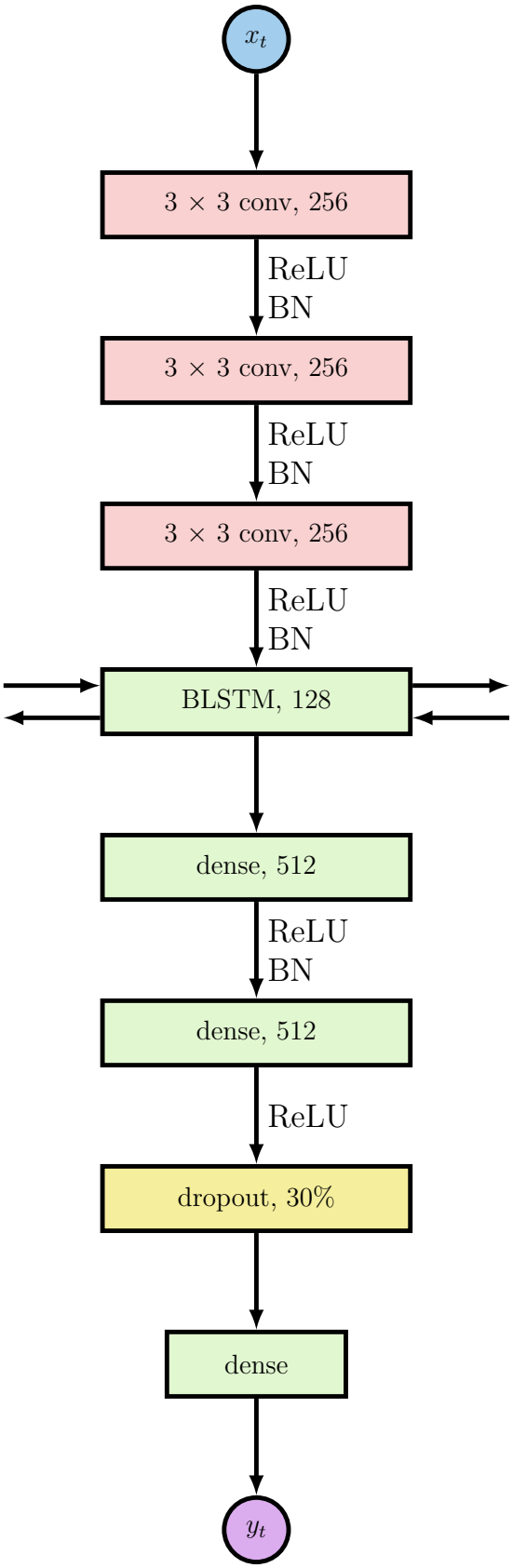

Figure 3.11: C-BLSTM architecture.

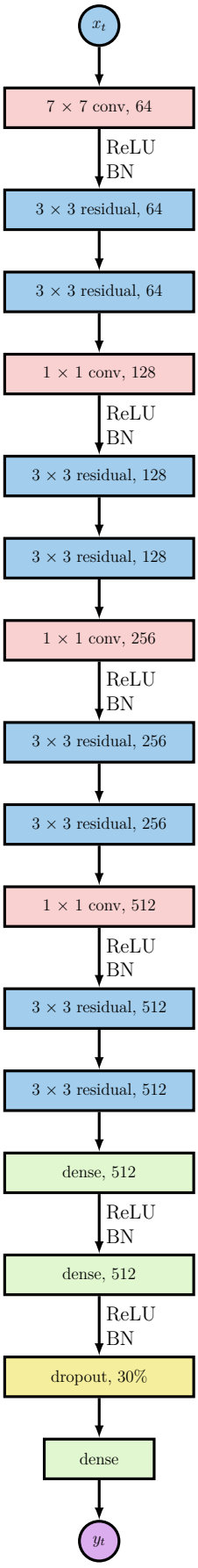

Figure 3.12: Modified ResNet18 implementation based on ResNet [14].

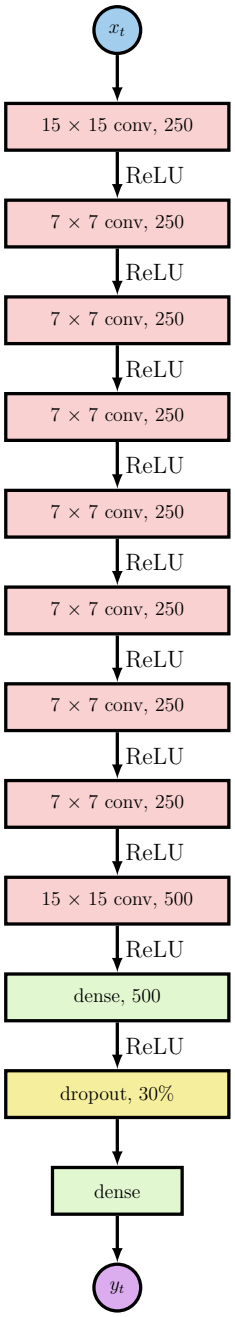

Figure 3.13: Simplified Wav2Letter architecture, based on Wav2Letter [6].

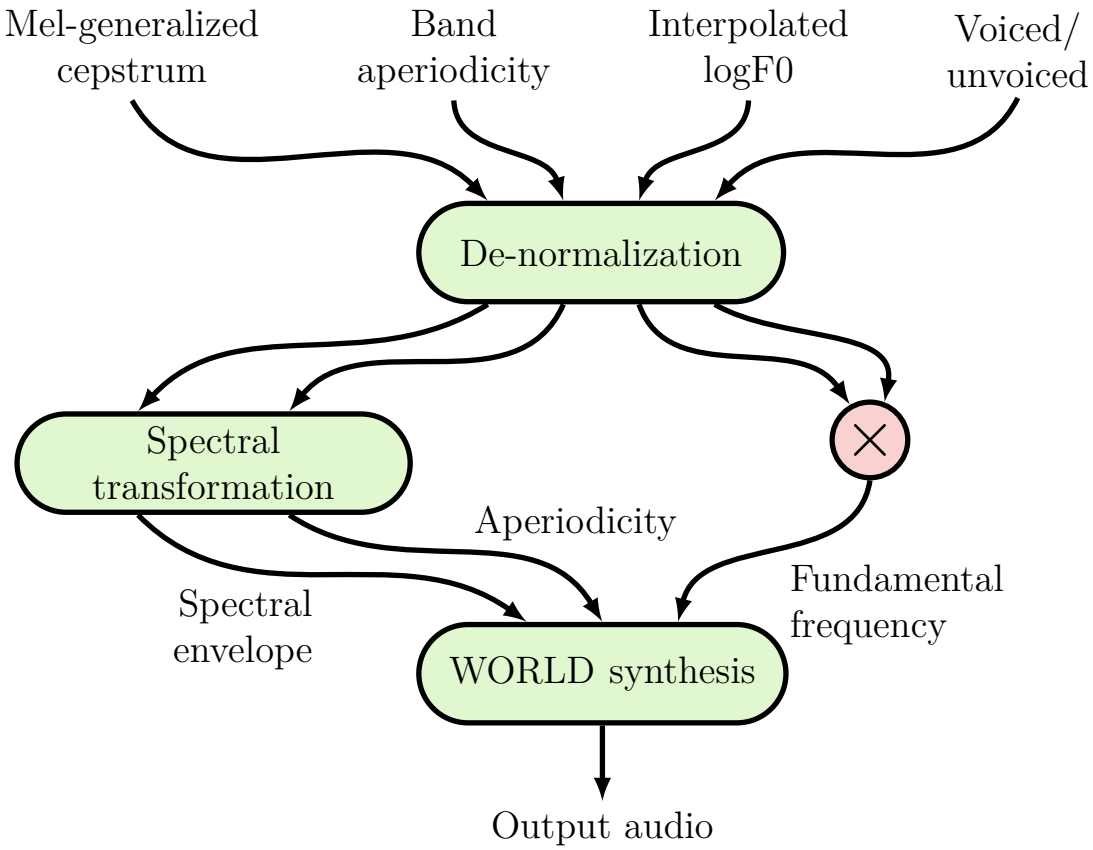

Figure 3.14: Post-processing pipeline.

3.7 Experimental Setup

The trainings were done on 2 computers, with the following specifications:

- Computer #1:
  - CPU: Intel® Core™ i7-6700 CPU, 3.40GHz (4 cores, 8 threads),
  - RAM: 32 GB,
  - GPU: NVIDIA® GeForce™ GTX 1080 8GB RAM.
- Computer #2:
  - CPU: Intel® Core™ i5-4690K CPU, 3.50GHz(4 cores, 4 threads),
  - RAM: 32 GB,
  - GPU: 2× NVIDIA® GeForce™ GTX TITAN X 12GB RAM.

The batch size was set to 32 and 128 on Computer #1 and #2 respectively. During all experiments early stopping [27] was used with patience of 10 epochs. For each model Adam [21] optimizer was used with default parameters suggested by the authors:

- $\alpha = 0.001$,
- $\beta_1 = 0.9$,
- $\beta_2 = 0.999$,
- $\epsilon = 10^{-8}$.

The proposed method was implemented using Bash, Python 2.7 and Python 3.5, using the Keras 2.0.8 [5] deep learning framework and Tensorflow 1.3 [1] as its backend.

The following software and libraries was also used:

- For DTW: fastdtw 0.3.2[1].
- For measuring the accuracies: python-Levenshtein 0.12.0[2].
- Speech toolkits for various tasks:
  - SoundFile 0.10.1[3],
  - SPTK 3.9[4],
  - SoX v14.4.2[5],
  - sndfile-convert 1.0.25[6].
- For the experiments we used Festival 2.5[7], but during the TTS comparison we also tried out the followings:
  - SVOX pico,
  - MaryTTS[8],
  - eSpeak[9].

[1] https://pypi.python.org/pypi/fastdtw
[2] https://github.com/ztane/python-Levenshtein
[3] https://github.com/bastibe/SoundFile
[4] http://sp-tk.sourceforge.net/
[5] http://sox.sourceforge.net/
[6] http://www.mega-nerd.com/libsndfile/
[7] http://festvox.org/
[8] https://github.com/marytts/marytts
[9] http://espeak.sourceforge.net/

- During the experiments pyworld 0.2.4[10] was used, which is a wrapper for WORLD [24], but several other was also tested:
  - Ahocoder 0.99[11],
  - MagPhase[12],
  - PulseModel[13].
- ASR systems:
  - Google Speech API[14],
  - Wav2Letter[15].

[10] https://github.com/JeremyCCHsu/Python-Wrapper-for-World-Vocoder
[11] http://aholab.ehu.es/ahocoder/index.html
[12] https://github.com/CSTR-Edinburgh/magphase
[13] https://github.com/gillesdegottex/pulsemodel
[14] https://cloud.google.com/speech/
[15] https://github.com/facebookresearch/wav2letter

Chapter 4

Results

This section describes the test results of the trained deep neural networks. Before training on larger datasets, a detailed sanity check was evaluated on TIMIT using dense, CNN and C-BLSTM architectures to make sure of the correct model responses on varying size of datasets. The second part of the chapter includes several figures showing details of training procedures. Learning curves are presented for each benchmark dataset per model architecture.

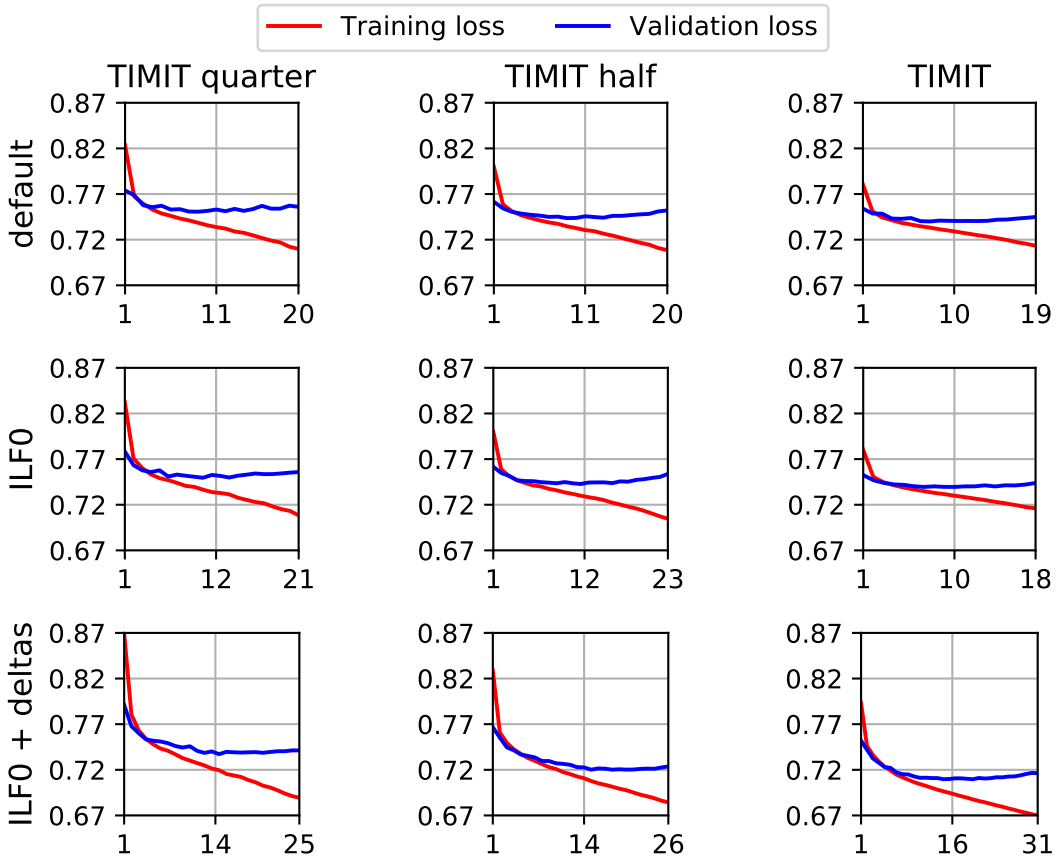

Figure 4.1: Sanity check on TIMIT subsets with the dense architecture. Each figure represent a learning curve, where x and y axis is the number of epochs and the loss respectively. With each subsequent column we double the size of the database and every row uses a different input and output feature pairs. As we double the size of the databases, the loss becomes smaller and smaller and by concatenating new features we can improve it even further. But the network won't converge.

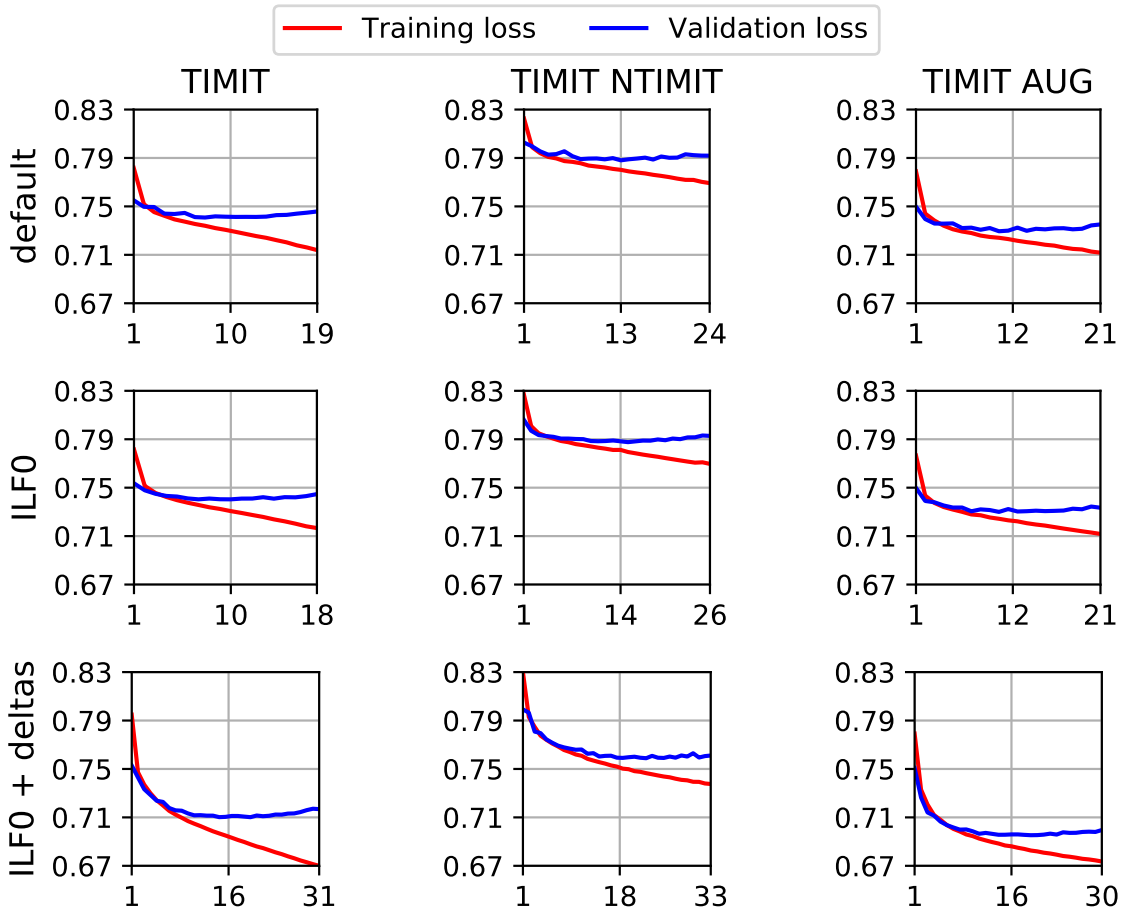

Figure 4.2: Training on TIMIT with the dense architecture. To further increase the size of the database we concatenated the TIMIT and the NTIMIT (2nd column), but with these new noisy samples the overall quality of the system has decreased. The augmentation (Section 3.5) of the TIMIT corpus (last column) only produced a slight improvement in the losses.

Used features		**TIMIT quarter**		**TIMIT half**		**TIMIT**		**TIMIT NTIMIT**		**TIMIT AUG**	
		war	lar	war	lar	war	lar	war	lar	war	lar
default	mean	43%	70%	45%	73%	52%	77%	49%	75%	50%	76%
	std	29%	20%	29%	19%	28%	17%	28%	18%	29%	19%
ILF0	mean	62%	83%	63%	84%	64%	85%	62%	84%	65%	85%
	std	28%	16%	27%	15%	28%	15%	29%	15%	27%	15%
IFL0 + deltas	mean	62%	83%	63%	84%	63%	84%	62%	84%	62%	84%
	std	28%	15%	26%	15%	28%	16%	27%	15%	27%	15%

Table 4.1: Accuracy of the dense architecture on the TIMIT corpus [42]. In this table we can see the ASR evaluated results of the previously (Figure 4.1 and 4.2) seen dense model. In case of the mean higher and for the standard deviation (std) lower is the better.

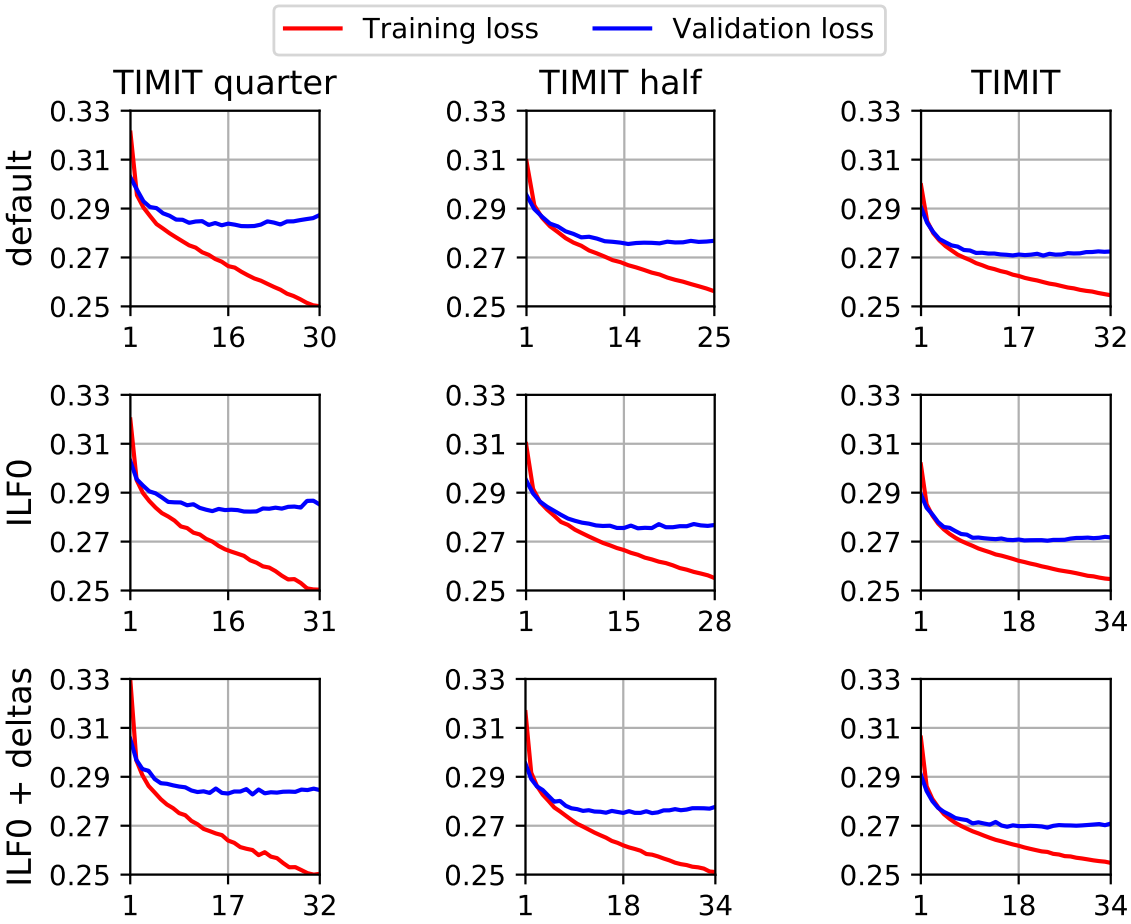

Figure 4.3: Sanity check on TIMIT with the CNN architecture. The notations are similar to Figure 4.1. Like before with increasing the dataset the loss decreases, but by introducing new features we cannot improve the results. The scale is differ from the previous because with CNNs we cannot mask the results.

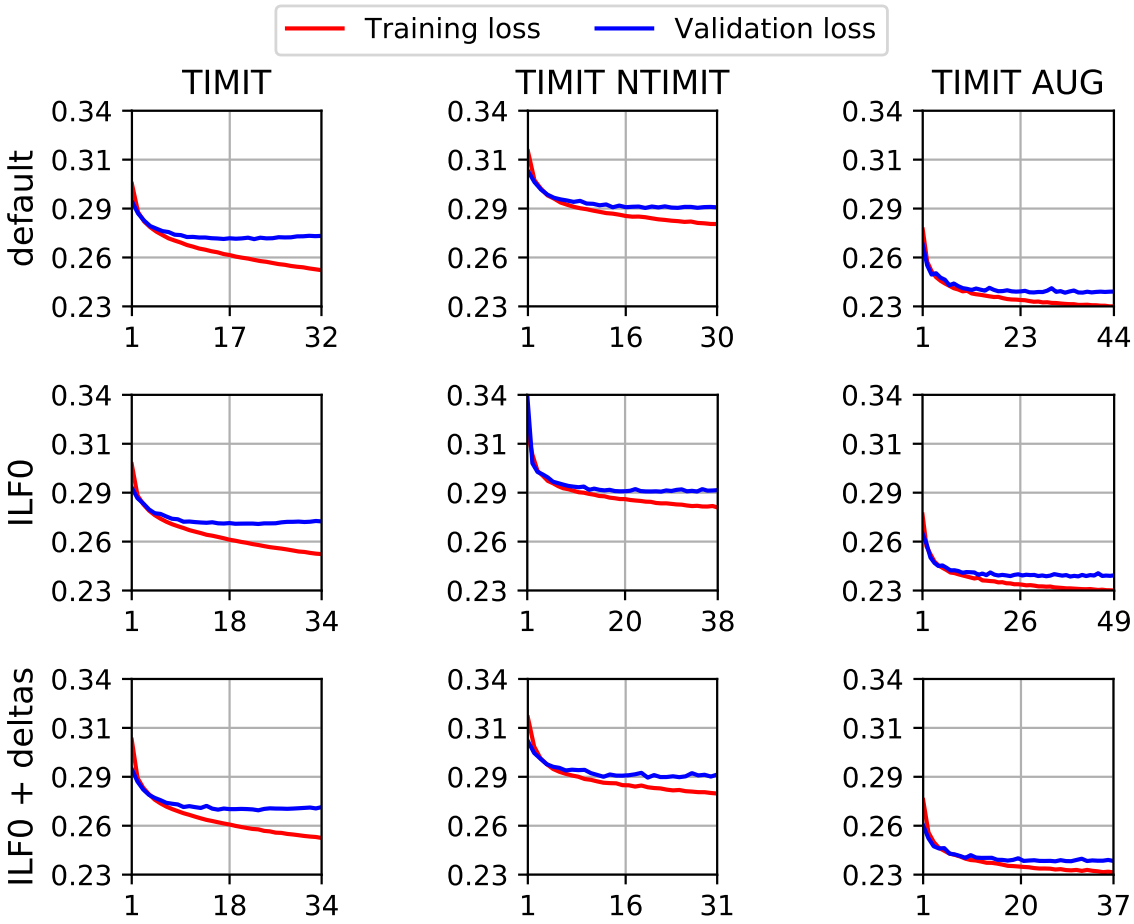

Figure 4.4: Training on TIMIT with the CNN architecture. As before the concatenated database worsened the results, but with the augmentation they became better.

Used features		**TIMIT quarter**		**TIMIT half**		**TIMIT**		**TIMIT NTIMIT**		**TIMIT AUG**	
		war	lar	war	lar	war	lar	war	lar	war	lar
default	mean	43%	71%	46%	74%	50%	76%	45%	73%	51%	77%
	std	29%	21%	27%	18%	28%	17%	28%	19%	29%	17%
ILF0	mean	51%	76%	58%	81%	59%	82%	57%	81%	60%	82%
	std	29%	20%	27%	16%	29%	17%	28%	16%	27%	15%
ILF0 + deltas	mean	52%	77%	54%	79%	60%	82%	59%	82%	59%	82%
	std	27%	18%	26%	15%	27%	15%	26%	14%	27%	15%

Table 4.2: Accuracy of the CNN architecture on the TIMIT corpus [42]. The notations are similar to Table 4.1.

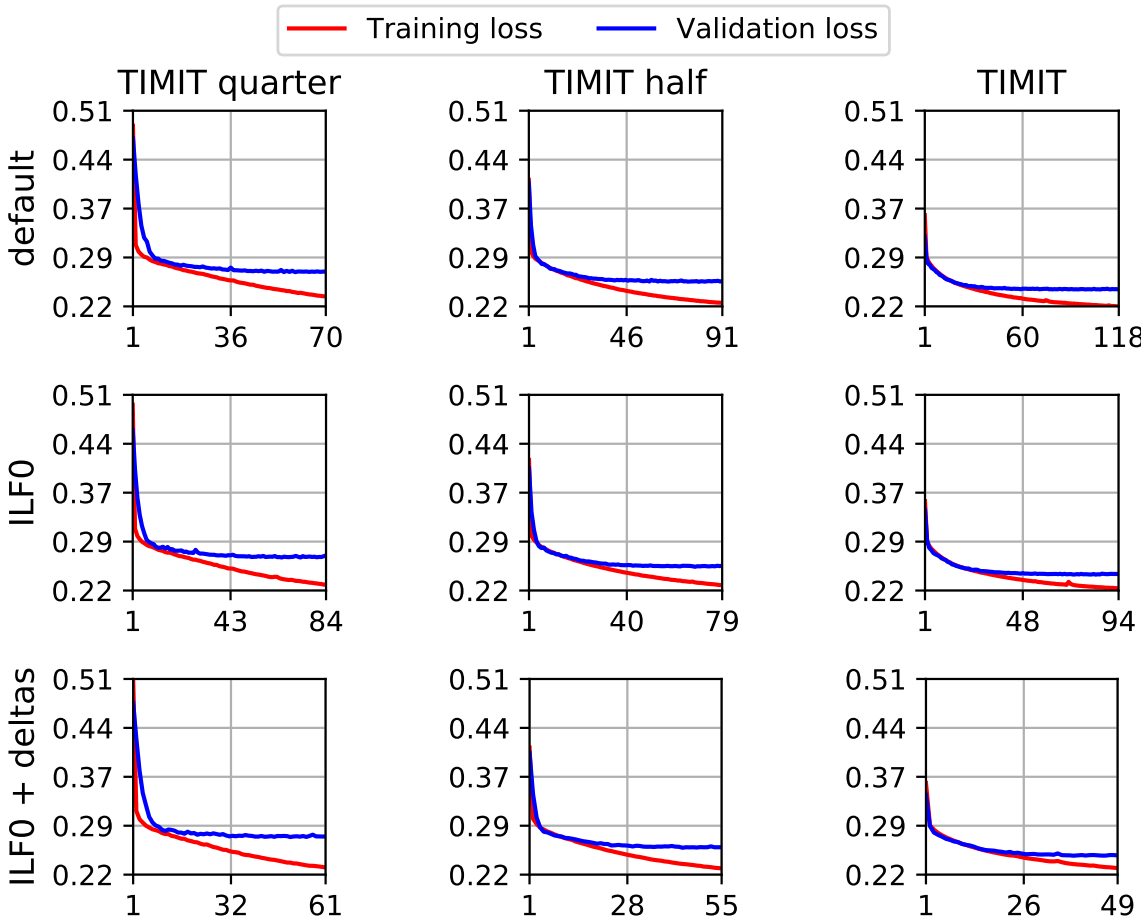

Figure 4.5: Sanity check on TIMIT with the C-BLSTM architecture. The notations are similar to the previous figures. The size of the database here also matters, but by using new features the accuracy won't improve, instead it cuts down the training time.

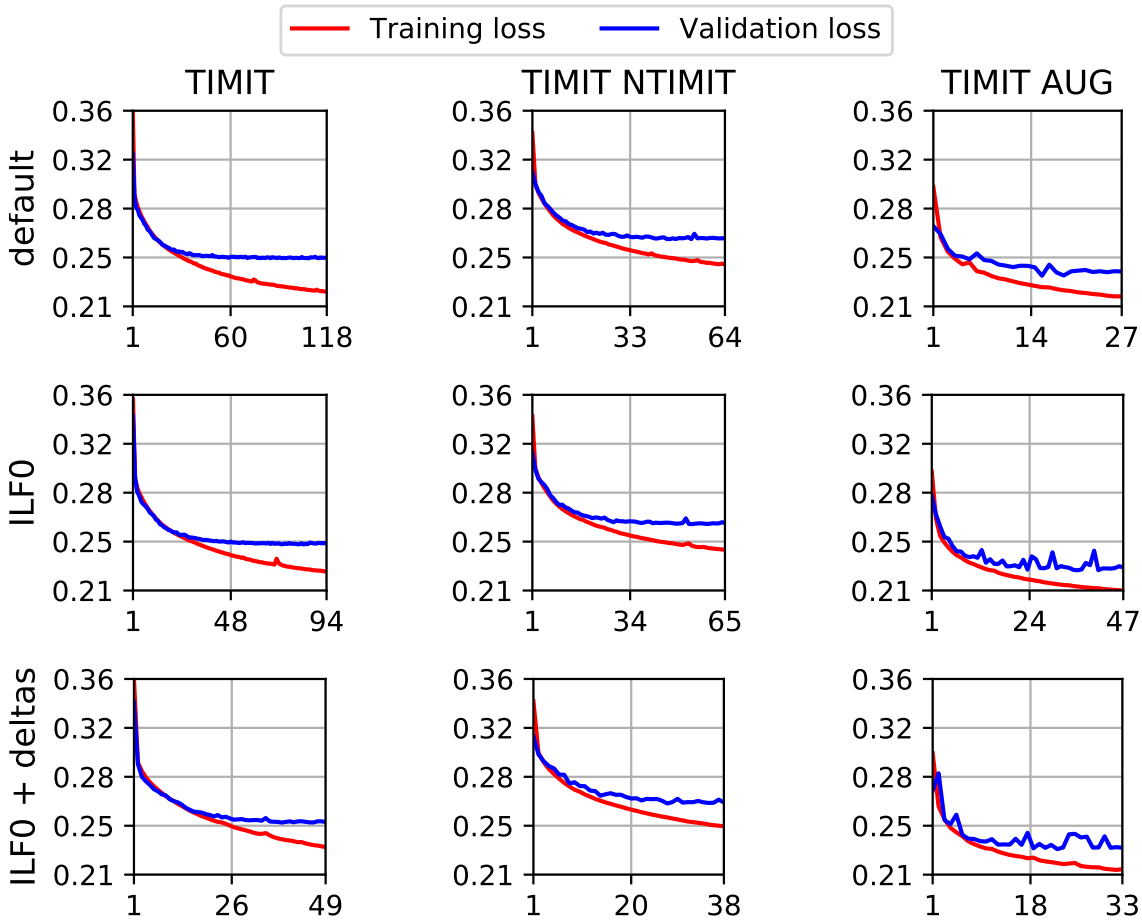

Figure 4.6: Training on TIMIT with the C-BLSTM architecture. It produced similar results as the one can be seen on Figure 4.2. Because of the batch normalization, small peaks appeared on the validation losses, but the losses were continuously improving.

Used features		**TIMIT quarter**		**TIMIT half**		**TIMIT**		**TIMIT NTIMIT**		**TIMIT AUG**	
		war	lar	war	lar	war	lar	war	lar	war	lar
default	mean	41%	70%	42%	72%	49%	77%	47%	75%	40%	70%
	std	27%	16%	27%	17%	26%	14%	27%	16%	27%	17%
ILF0	mean	39%	70%	49%	76%	54%	79%	51%	78%	53%	79%
	std	27%	17%	28%	16%	26%	15%	29%	17%	27%	15%
ILF0 + deltas	mean	38%	70%	51%	78%	54%	79%	47%	76%	53%	79%
	std	28%	18%	26%	16%	27%	15%	27%	15%	28%	15%

Table 4.3: Accuracy of the C-BLSTM architecture on the TIMIT corpus [42].

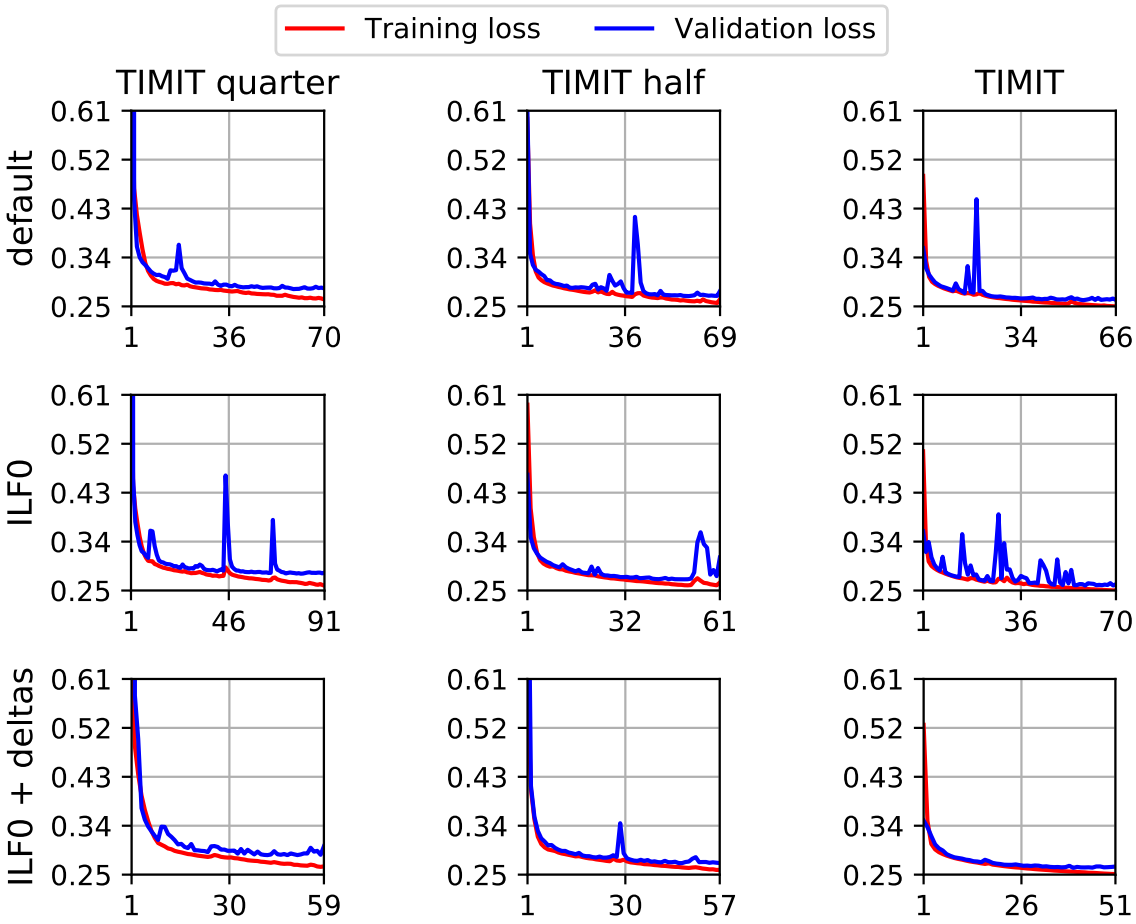

Figure 4.7: Sanity check on TIMIT with the ResNet architecture without the last residual layers.

Used features		**TIMIT quarter**		**TIMIT half**		**TIMIT**	
		war	lar	war	lar	war	lar
default	mean	43%	72%	48%	75%	53%	78%
	std	26%	17%	29%	17%	29%	15%
ILF0	mean	46%	74%	52%	78%	59%	82%
	std	28%	18%	28%	16%	27%	15%
ILF0 + deltas	mean	46%	75%	57%	81%	57%	81%
	std	28%	17%	27%	15%	27%	14%

Table 4.4: Accuracy of the ResNet architecture without the last residual layers on the TIMIT corpus.

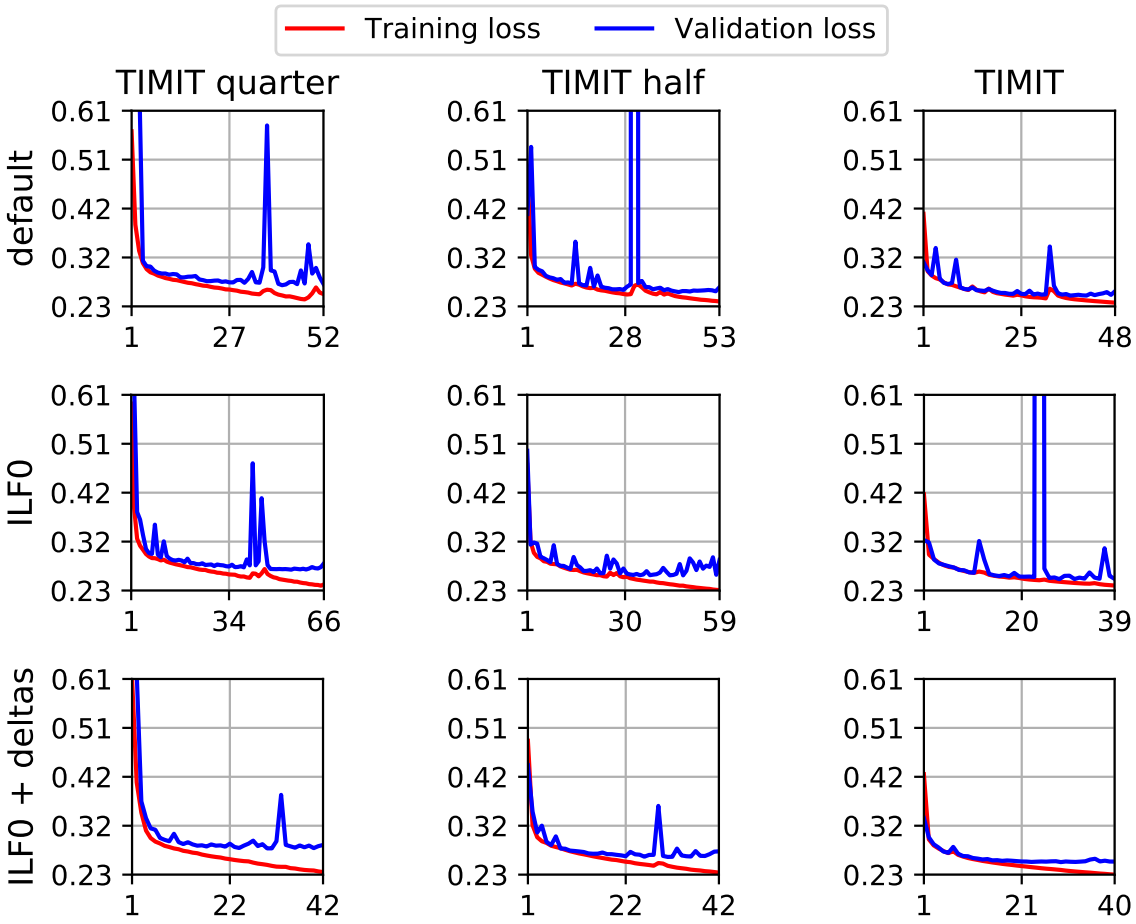

Figure 4.8: Sanity check on TIMIT with the ResNet architecture without the last residual layers but with bigger kernels (first layer: 15, others: 7).

Used features		**TIMIT quarter**		**TIMIT half**		**TIMIT**	
		war	lar	war	lar	war	lar
default	mean	38%	69%	48%	76%	54%	79%
	std	26%	17%	28%	15%	28%	14%
ILF0	mean	44%	72%	52%	78%	57%	81%
	std	26%	17%	27%	15%	28%	15%
ILF0 + deltas	mean	39%	69%	51%	78%	54%	80%
	std	28%	19%	27%	16%	27%	14%

Table 4.5: Accuracy of the ResNet architecture without the last residual layers but with bigger kernels (first layer: 15, others: 7) on the TIMIT corpus.

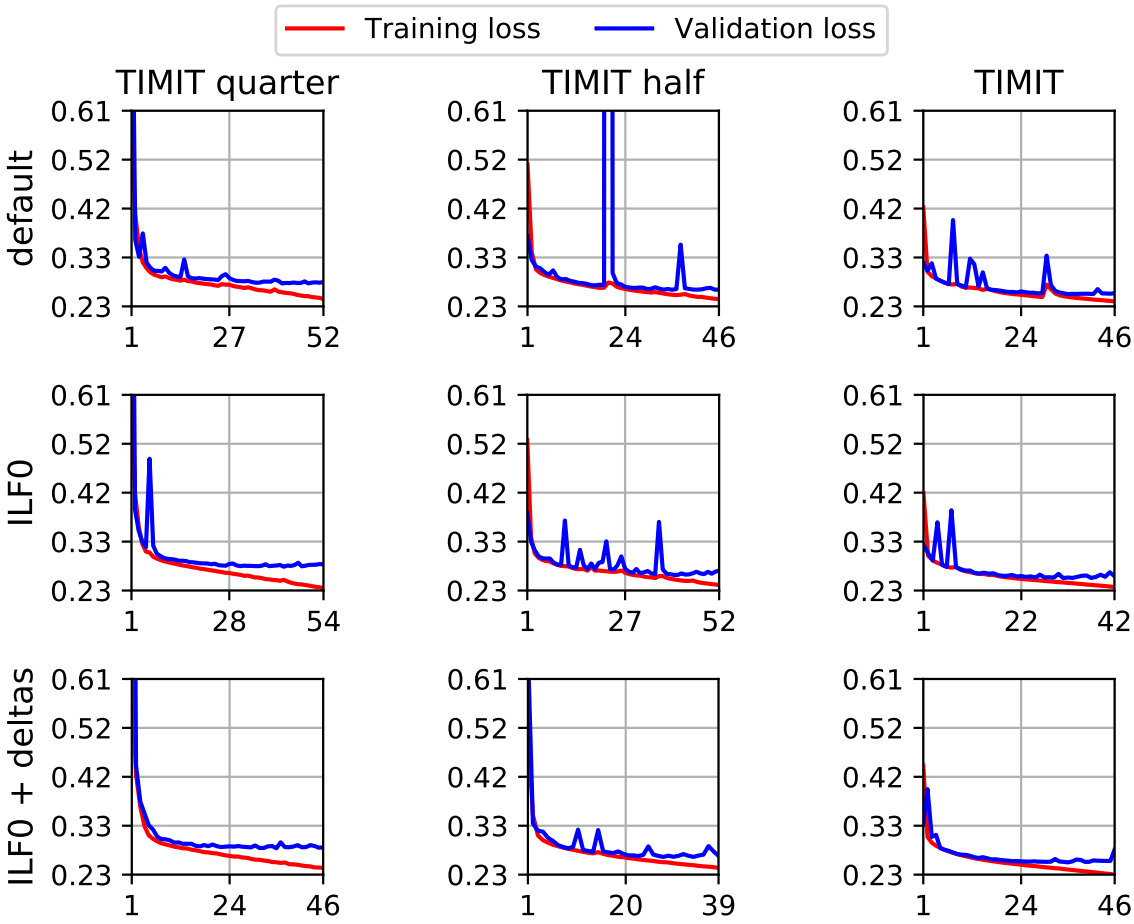

Figure 4.9: Sanity check on TIMIT with the ResNet18 architecture.

Used features		**TIMIT quarter**		**TIMIT half**		**TIMIT**	
		war	lar	war	lar	war	lar
default	mean	41%	70%	50%	76%	54%	79%
	std	29%	18%	28%	16%	28%	15%
ILF0	mean	42%	71%	56%	80%	58%	82%
	std	27%	20%	27%	17%	27%	16%
ILF0 + deltas	mean	45%	74%	54%	79%	59%	82%
	std	27%	16%	28%	15%	27%	14%

Table 4.6: Accuracy of the ResNet18 architecture on the TIMIT corpus.

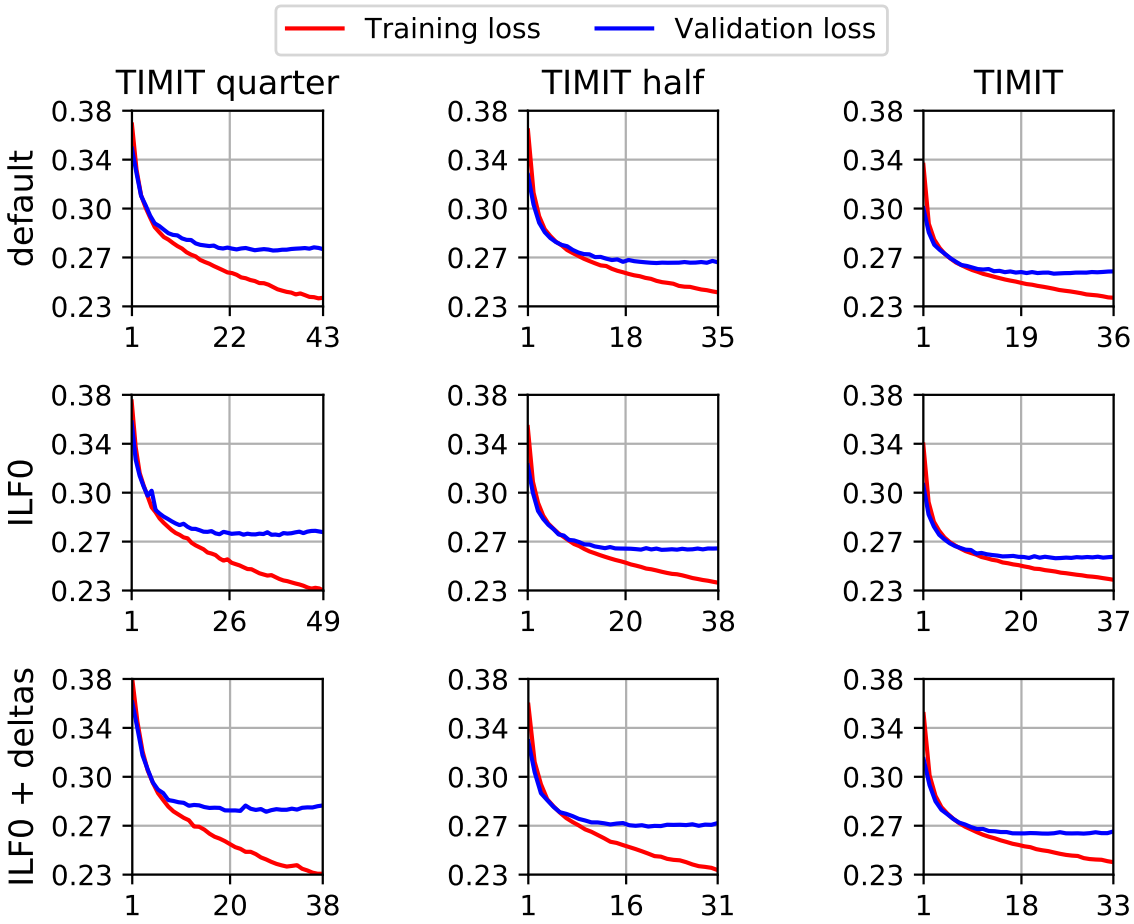

Figure 4.10: Sanity check on TIMIT with the Wav2Letter architecture.

Used features		**TIMIT quarter**		**TIMIT half**		**TIMIT**	
		war	lar	war	lar	war	lar
default	mean	34%	67%	42%	72%	49%	76%
	std	26%	16%	28%	17%	28%	16%
ILF0	mean	40%	71%	48%	76%	55%	79%
	std	27%	16%	28%	16%	29%	17%
ILF0 + deltas	mean	32%	65%	43%	73%	52%	77%
	std	25%	18%	28%	17%	27%	16%

Table 4.7: Accuracy of the Wav2Letter architecture on the TIMIT corpus.

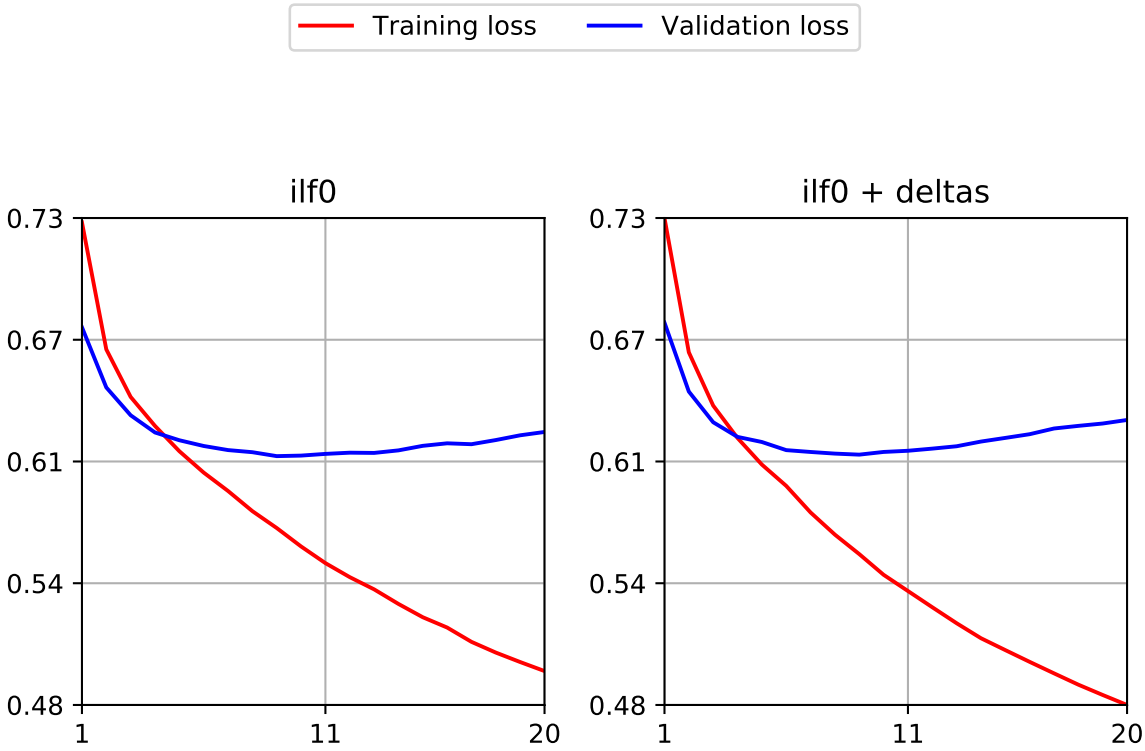

Figure 4.11: Training on TIMIT with the BLSTM architecture. After a few iteration it started to overfit regardless of the features.

Used features		**CSLU KIDS half**		**CSLU KIDS**		**CSLU KIDS filtered**		**CSLU KIDS filtered AUG**	
		war	lar	war	lar	war	lar	war	lar
Dense	mean	36%	49%	35%	50%	47%	43%	36%	49%
ILF0	std	47%	44%	48%	43%	47%	43%	47%	43%
CNN	mean	24%	38%	34%	50%	39%	56%	34%	48%
ILF0	std	41%	40%	46%	41%	49%	40%	46%	41%

Table 4.8: Accuracies on the CSLU Kids' Speech corpus.

Used features		**CSLU KIDS on TNT**		**CSLU KIDS on TNT AUG**	
		war	lar	war	lar
Dense	mean	32%	44%	32%	46%
ILF0	std	46%	43%	46%	42%
CNN	mean	33%	50%	33%	49%
ILF0	std	46%	40%	46%	41%

Table 4.9: Accuracies on the CSLU Kids' corpus using transfer learning. TNT is an abbreviation of TIMIT_NTIMIT for the sake of simplicity.

4.1 Optimizing CNN architecture

In Section 2.9, hyper-parameter optimization methods were introduced. The learning curves of TIMIT dataset show an early overfitting to the training data. The search for a better neural network architecture is computed with the Tree-structured Parzen Estimator algorithm. A well-known TPE implementation is Hyperopt[1]. Only convolutional neural networks were used during hyper-parameter optimization.

The first assumption was that a smaller network might not be prone to overfitting. The following hyper-parameters were tuned during the first set of trials:

Hyper-parameters	**Values**
Number of convolutional layers	[1, 2, 3]
Number of hidden units per convolutional layer	[32, 64, 128, 256, 512]
Kernel size	[3, 5, 7, 10]
Number of dense layers	[1, 2, 3]
Number of hidden units per dense layer	[64, 128, 256, 512, 1024]
Dropout rate	$0 \leq x \leq 0.75$

Table 4.10: Hyper-parameter space for 54 trials.

Based on the learning curves of ConvNets, it was reasonable to limit the training procedure to the first 15 epochs. With the applied restriction the overall training time requirement was reduced drastically. 54 models were trained using two NVIDIA® GeForce™ GTX TITAN X for 27 hours.

During the TPE warm-up phase, 20 trials were evaluated with random configurations. Then another 34 model was trained as candidate configurations of TPE. However, the results showed the opposite: the highest values for almost every hyper-parameter were chosen to achieve the best validation loss with 0.264. The best model architecture consist of three 1D convolutional layers (256 hidden units each), followed by another three dense layers (1024 hidden neurons). Kernel size was set to 5 and the dropout rate to 0.19.

For this reason, the parameter space was widened:

[1] http://jaberg.github.io/hyperopt/

Hyper-parameters	Values
Number of convolutional layers	[1, 2, 3, 4, 5]
Number of hidden units per convolutional layer	[32, 64, 128, 256, 512, 1024]
Kernel size	[3, 5, 7, 10, 13]
Number of dense layers	[1, 2, 3, 4, 5]
Number of hidden units per dense layer	[64, 128, 256, 512, 1024, 2048]
Dropout rate	$0 \leq x \leq 0.75$

Table 4.11: Hyper-parameter space for another 16 trials.

Using the widened parameter space 16 more candidate models were trained. The procedure took 9 hours. TPE chose the subsequent trials based on its history. Comparison of the validation losses of the trials can be seen on Figure 4.12. The best model configuration of the first 54 trials is plotted in orange, while the best option from 70 models is shown in red color.

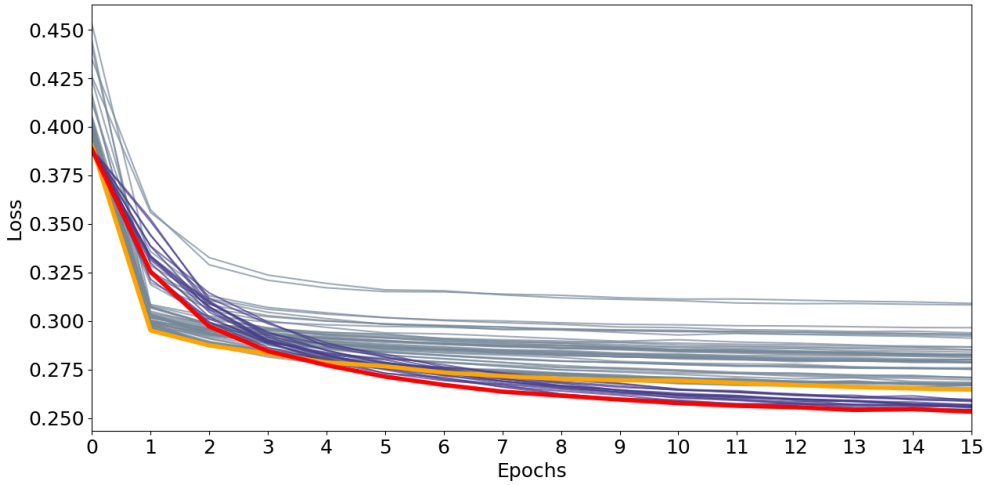

Figure 4.12: Validation losses of 70 different convolutional neural networks during hyper-parameter optimization. Grey: first set of trials (54 models were trained). Blue: second set of trials (another 16 models were trained considering the information gathered from the first set of trials). Orange: Validation loss of the best model of the first set during the first 15 epochs. Red: Validation loss of the best model of the full set during the first 15 epochs.

Figure 4.13 shows that with better hyper-parameters, noticeable improvement can be achieved. The best candidate model in 70 trials include five convolutional and four dense layers (512 and 2048 hidden units respectively). The kernel size was set to 13, and the dropout rate was 0.42. A ConvNet with this configuration achieved 0.254 validation error during training on the TIMIT dataset. Regardless of a minimal validation loss improvement, the trained model proposed by TPE is achieved the exact same letter accuracy rate calculated by the Google Speech API

on the the test set. We tried to train the model from the 15^{th} epoch with the CSLU Kid's Speech corpus using principles of transfer learning. The model trained a few epochs, but it cannot fit the problem well. The result of this experiment was a 25% word and a 46% letter accuracy on the test set, with 42% and 36% standard deviation respectively. The reason of relatively low score and high deviation is mentioned in the previous section.

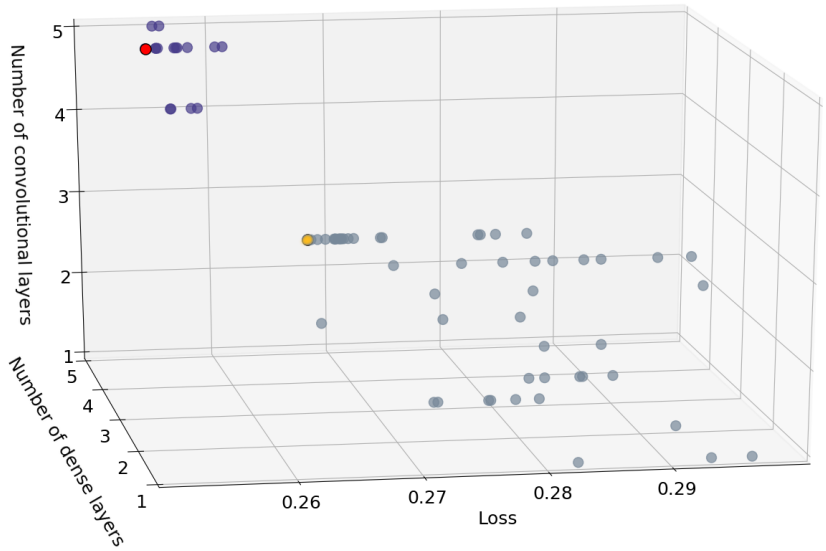

Figure 4.13: Grey markers: trials from the first parameter space. Blue markers: trials from the second, widened parameter space considering the previous evaluations. Orange marker: Best model of the first set based on validation loss. Red marker: Best model out of the 70 examined configurations based on the same measure.

4.2 Human evaluation

We measured the success of de-identification by conducting a human estimation experiment. Twenty samples were selected from the synthesized outputs of the anonymization process. The audio files contained speech segments of adults and children as well. The following question was asked: which samples were produced by kids?

Fourteen people were participated in the following human estimation experiment (Figure 4.14). The overall responses are satisfying. The participants cannot deduced the origin of the speaker. Based on the results, the answers seem like random guessing. There are a few outliers: the speakers of sound files contained the “awesome” and “potatoes” words were guessed as children the most confidently. In this case, the participants linked together the easiest, and most common words with the children, nevertheless the "potatoes" word was said by an adult. Towards the other end of the scale, the hardly pronounceable words were linked to adult speakers (“tradition” and “visually”).

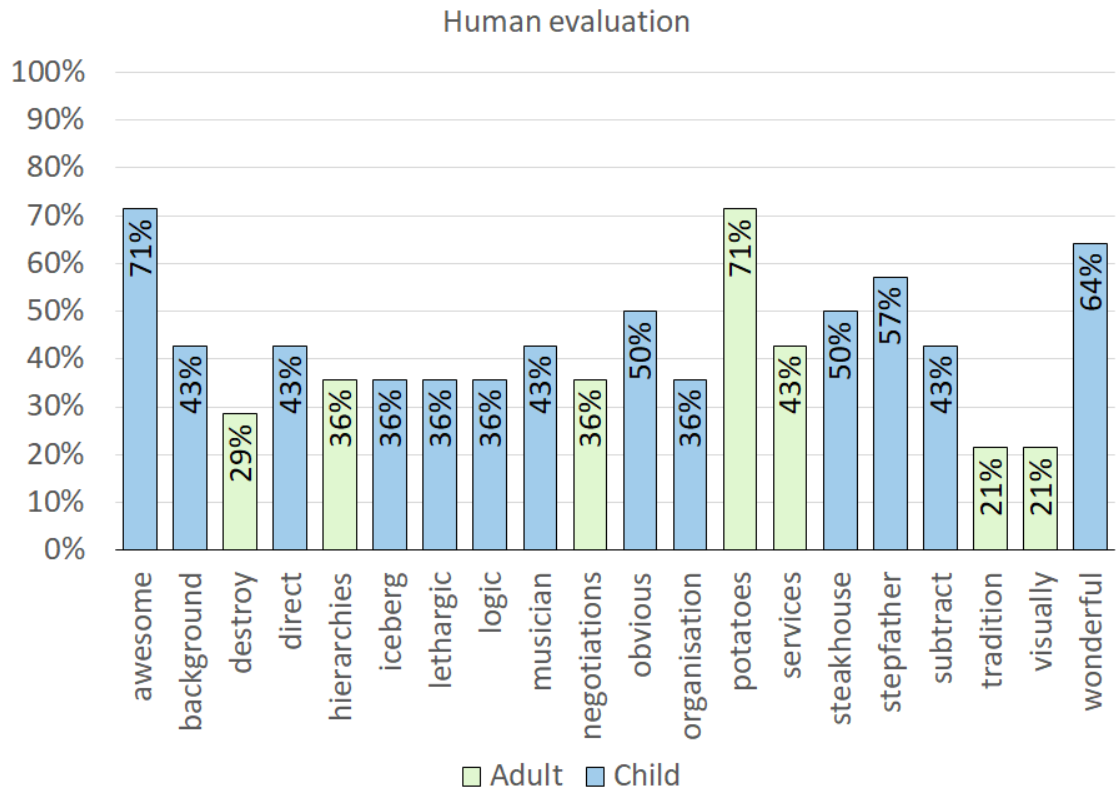

Figure 4.14: Participants’ answers in case of 20 audio files for the following question: “Which samples were produced by kids?”. The speech signals produced by adults are indicated with green color. Blue color marks the samples said by children.

Chapter 5

Discussion

The sanity checks proved the assumption that increasing the size of training set also improves the accuracy on the validation and test sets. However, the applied speed perturbation technique did not give rise to further betterment.

The tested deep neural networks performed well on the TIMIT dataset: they mostly achieved 80-85% letter error rate that yield understandable synthesized voice.

As we can see the results on the CSLU Kids' Speech had much poorer quality compared to the ones trained on TIMIT (Table 4.8). The quality of the transformation is noisy, and two main cases are distinguished based on the lower accuracy and higher deviation values:

- the original samples are hard to understand, and therefore the transformation is not as good as expected: this case results in near 0 war/lar in most cases;
- the other case is that the speech segments of the children are audible and clear, so the automatic speech recognition system transcribes the anonymized audio correctly, which results higher war/lar values.

We experimented with transfer learning as well. A pretrained TIMIT_NTIMIT models was used to enhance model capability during training on the Kids' database. Table 4.9 shows, that the CNN architecture outperformed the dense one, and produced slightly better output. After loading the TIMIT_NTIMIT model the weights of the last dense layer were replaced with random values. None of the the previous layers were fixed during fine-tuning, because of the small network architecture. The training time drastically reduced, and the training resulted an 50% better accuracy in case of CNN on the highly filtered Kids' subset. The augmentation brings extra noise, in addition to the gain, however the differences are insignificant.

The conclusion of the hyper-parameter optimization is that deep architectures with carefully chosen hyper-parameters can improve the training procedure, however, a smaller network can model the voice conversion assignment considerably.

Chapter 6

Conclusions

In this thesis we presented a speech de-identification method, that can anonymize any speaker's voice with minimal loss of the overall quality in term of further processability, even when simple models, like the dense architecture is used.

The method we introduced is far from perfect, there are many things, that could be easily improved. We could change many parts of the system into a more accurate one, but these usually are commercials ones. A neural vocoder could be trained to achieve a representation that is easier to learn. The training time could be drastically reduced with Truncated Back Propagation Through Time [39] or with synthetic gradients [17]. By using safe words to replace sensitive ones and with simple speech processing transformations, the degree of de-identification could easily be improved without any loss.

The proposed method can be used anywhere, where the privacy of the user is crucial, for example in cloud-based speech services or medical records. Each of the architectures that we used is common and easy to train, because of this they can be used even on mobile devices with some simple modification.

Chapter 7

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[3] James Bergstra, Yoshua Bengio, Bardenet Remi, and Balázs Kégl. Algorithms for hyper-parameter optimization. *25th Annual Conference on Neural Information Processing Systems*, pages 2546–2554, 2011.

[4] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[5] François Chollet et al. Keras. `https://keras.io`, 2015.

[6] Ronan Collobert, Christian Puhrsch, and Gabriel Synnaeve. Wav2letter: an end-to-end convnet-based speech recognition system. *CoRR*, abs/1609.03193, 2016.

[7] Gilles Degottex, Pierre Lanchantin, Mark Gales, Gilles Degottex, Pierre Lanchantin, and Mark Gales. A log domain pulse model for parametric speech synthesis. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 26(1):57–70, 2018.

[8] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[9] Daniel Erro, Iñaki Sainz, Eva Navas, and Inma Hernaez. Harmonics plus noise model based vocoder for statistical parametric speech synthesis. *IEEE Journal of Selected Topics in Signal Processing*, 8(2):184–194, 2014.

[10] Felipe Espic, Cassia Valentini-Botinhao, and Simon King. Direct modelling of magnitude and phase spectra for statistical parametric speech synthesis. *Proc. Interspeech, Stochohlm, Sweden*, 2017.

[11] Toshiaki Fukada, Keiichi Tokuda, Takao Kobayashi, and Satoshi Imai. An adaptive algorithm for mel-cepstral analysis of speech. In *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, volume 1, pages 137–140. IEEE, 1992.

[12] Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. *Proceedings of the 31st International Conference on Machine Learning (PMLR)*, 32:1764–1772, 2014.

[13] Awni Y. Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. Deep speech: Scaling up end-to-end speech recognition. *CoRR*, abs/1412.5567, 2014.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural-computation*, 9:1735–80, 1997.

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[17] Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. *arXiv preprint arXiv:1608.05343*, 2016.

[18] Navdeep Jaitly and Geoffrey E. Hinton. Vocal tract length perturbation (vtlp) improves speech recognition. *Proceedings of the 30th International Conference on Machine Learning*, 28, 2013.

[19] Charles Jankowski, Ashok Kalyanswamy, Sara Basson, and Judith Spitz. Ntimit: A phonetically balanced, continuous speech, telephone bandwidth speech database. In *Acoustics, Speech, and Signal Processing, 1990. ICASSP-90., 1990 International Conference on*, pages 109–112. IEEE, 1990.

[20] Hideki Kawahara, Ikuyo Masuda-Katsuse, and Alain De Cheveigne. Restructuring speech representations using a pitch-adaptive time–frequency smoothing and an instantaneous-frequency-based f0 extraction: Possible role of a repetitive structure in sounds1. *Speech communication*, 27(3-4):187–207, 1999.

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[22] Tom Ko, Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur. Audio augmentation for speech recognition. *Proc. Interspeech, Dresden, Germany*, 2015.

[23] Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron Courville, and Yoshua Bengio. Samplernn: An unconditional end-to-end neural audio generation model. *arXiv preprint arXiv:1612.07837*, 2016.

[24] Masanori Morise, Fumiya Yokomori, and Kenji Ozawa. World: a vocoder-based high-quality speech synthesis system for real-time applications. *IEICE TRANSACTIONS on Information and Systems*, 99(7):1877–1884, 2016.

[25] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 5206–5210. IEEE, 2015.

[26] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. The kaldi speech recognition toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, December 2011. IEEE Catalog No.: CFP11SRW-USB.

[27] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998.

[28] Stan Salvador and Philip Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561–580, 2007.

[29] M. Schuster and K. K. Paliwa. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[30] Diemo Schwarz. *Spectral envelopes in sound analysis and synthesis*. PhD thesis, Universität Stuttgart, Fakultät Informatik, 1998.

[31] Khaldoun Shobaki, John-Paul Hosom, and Ronald A. Cole. The ogi's kid's speech corpus and recognizers. *Proc. 6th International Conference on Spoken Language Processing (ICSLP 2000), Beijing, October*, 2000.

[32] Khaldoun Shobaki, John-Paul Hosom, and Ronald A. Cole. Cslu: Kids' speech version 1.1, 2007. Linguistic Data Consortium (LDC) catalog number LDC2007S18 and isbn 1-58563-395-X, `https://catalog.ldc.upenn.edu/LDC2007S18`.

[33] Jose Sotelo, Soroush Mehri, Kundan Kumar, Joao Felipe Santos, Kyle Kastner, Aaron Courville, and Yoshua Bengio. Char2wav: End-to-end speech synthesis. 2017.

[34] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[35] Keiichi Tokuda, Takao Kobayashi, Takashi Masuko, and Satoshi Imai. Mel-generalized cepstral analysis-a unified approach to speech spectral estimation. In *Third International Conference on Spoken Language Processing*, 1994.

[36] Aaron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[37] Gabriel Synnaeve Vitaliy Liptchinsky and Ronan Collobert. Letter-based speech recognition with gated convnets. *CoRR*, abs/1712.09444, 2017.

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

[39] Ronald J Williams and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural computation*, 2(4):490–501, 1990.

[40] Zhizheng Wu, Oliver Watts, and Simon King. Merlin: An open source neural network speech synthesis system. *Proceedings of the 9th ISCA Speech Synthesis Workshop (SSW9)*, 2016.

[41] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[42] Victor Zue, Stephanie Seneff, and James Glass. Speech database development at mit: Timit and beyond. *Speech Communication*, 9:351–356, 1990.