
POLISH-JAPANESE ACADEMY OF INFORMATION TECHNOLOGY

**Computer Science Faculty**

**Multimedia Department**

Interaction Human-Computer

**Rafał Rolczyński**

Album No. 15497

# Synthetic Boosted Automatic Speech Recognition

**Master Thesis**

Prof. dr hab. Krzysztof Marasek

Warsaw, 25 September 2019

**Abstract**

Nowadays, speech recognition is an active research field, where various deep neural architectures are explored. The published successful models are optimized on massive transcribed datasets, which most of them are closed. A deep neural network solves two closely related tasks. It learns to recognize phonemes and to formulate grammar rules at the same time. In fact, a model is able to parallel and accurate build both of them, when a training corpus is large enough. However, inflected languages such as Polish contains much more grammar rules to define than in the case of English. Therefore, in order to achieve comparable results in the Polish language, the corpus must be substantially larger than the one presented for the English language. In contrast, to build more massive datasets, we present the *Synthetic Boosted Model*, which is an attempt to use synthetic data to enrich more profound the implicit language model. In the presented work, we propose the new model architecture, the new objective function, and the new training policy.

**Keywords**: automatic speech recognition, deep neural network, recurrent neural network, connectionist temporal classification, data augmentation, synthetic data, synthetic boosted

# Contents

# Chapter 1

# Introduction

Communication is a primary human need, and speaking is the most natural and fundamental form of communication complementary by eye contact, facial expressions, and body language. Even before the digital era, humans try to process speech information automatically [8, 9], what in consequence leads to form the *speech processing* domain. The *speech processing*, in general, can be divided into key components: *synthesis*, *recognition*, and *coding*. Among those, the recognition deals with basic information that speech provides such as language (*language identification*), message of words (*speech recognition*), and detailed information about the speaker, for instance, the speaker's emotions and gender (*speaker recognition*). Nowadays, the voice is becoming more and more often a point of interest of various information systems, including *intelligent assistance* in mobile phones, *in-car systems* and numerous documenting and monitoring systems (court, health service, or call center). Each application, where the voice is in interest, has the core component which is the *Automatic Speech Recognition* (ASR) system. The goal of ASR is to predict the correct transcription based on the given audio data.

The automatic speech processing began in the 1930s, although the first more widely used ASR systems appeared in the '70s [2, 9]. Initially, speech recognition systems could not recognize more than a few hundred words, and often required the breaks between words spoken in sequence. The accelerated development of computers in the '90s has resulted in several complex ASR systems based on *Hidden Markov Models* (HMM) [9]. In those days, speech recognition systems are composed of many stages, including handcrafted process of features extraction, and mentioned models HMM. Each of the components required fine-tuning, so in consequence, the entire process of building such system is time-consuming and required expertise. These systems are inflexible, so any change required the long re-adjustments.

Despite the significant improvements, the ASR systems could not compete with the

humans quality of speech recognition. A breakthrough in speech recognition is the application of *Deep Neural Networks* (DNN) at the beginning of the XXI century. In the first phase, deep learning algorithms play a limited role as acoustic models, or language models to rescore results [4, 5, 7]. The situation is changed when an automatic speech recognition system fully based on the *Recurrent Neural Network* (RNN) is introduced in 2014 [24]. Unlike traditional systems based on HMM, the entire process of automatic speech recognition is accomplished *end-to-end* by a single deep neural network. The key idea is to use the *Connectionist Temporal Classification* (CTC) algorithm [10], which enables either to train a model or to do inference. Next presented CTC based models are systems named *Deep Speech* and *Deep Speech 2* [25, 30], A huge dataset, a modified model architecture and an efficient optimization process enable *Deep Speech 2* to achieve impressive results for both English and Mandarin.

Nowadays, speech recognition is an active research field, where various deep neural architectures are explored. One of the remarkable architecture is the model *Listen, Attend and Spell* (LAS) [31], which is based on the Sequence-to-Sequence concept [23]. The LAS is the speech recognition system *end-to-end*, which is composed of an encoder, a decoder, and an attention mechanism [22, 31]. As the CTC based models, the Sequence-to-Sequence models also use an external static n-gram language model to correct minor language mistakes. Moreover, there are more sophisticated approaches, where an external language model is internally integrated [32, 38, 50]. The vanilla Sequence-to-Sequence models, due to the working principle of the encoder-decoder pair, require a priori the whole sequence of input data to be able to perform a prediction. Therefore, the use of the *Sequence-to-Sequence* models, in particular during online inferences, can be more difficult.

The success of the presented models is based on a large amount of data and an infrastructure which is capable of optimizing the model in a reasonable time of several days. The aforementioned models have been trained on data sets ranging from a few to tens of thousands of hours. Unfortunately, access to the data is severely limited. The vast majority of transcribed audio datasets are closed, for less popular languages in particular. Furthermore, obtaining a manual transcription of audio corpora is laborious and costly, and furthermore, beside efforts, a high human transcribing error rate persists. As a result, the development of both research and new commercial solutions, in speech recognition is limited, even though the weakly supervised and the unsupervised methods are explored [41, 44, 52]

The training dataset can be extended using synthetic audio data generated by *Text-to-Speech* models. This approach with success is a widely used technique of data augmentation [46, 47, 53, 54]. Unfortunately, the improvement exists only up to the ratio

1 to 1, rich training data to synthetic data. Adding more synthetic data to a training dataset has the opposite effect, and the efficiency of the system declines. In this thesis, we hypothesize that ASR systems can benefit from much larger synthesized corpora. We present the *Synthetic Boosted Model*. The model uses the synthetic data to enrich the language information thanks to the new model architecture, the new objective function, and the new training policy.

## 1.1 Thesis Outline

The thesis is organized into 8 chapters, which are described as follows:

**Chapter 1** The current chapter provides general information about our research interest, speech recognition, and its related context.

**Chapter 2** This chapter revises fundamental knowledge required to understand the key concepts presented in our work. Important topics are signal representations, recurrent neural networks, and the *Connectionist Temporal Classification*.

**Chapter 3** This chapter shows the available corpus, and describes how the data preprocessing is done. We analyze corpus with particular care before starting to explore different models. We introduce the audio representation, the dataset construction, and the data augmentation method.

**Chapter 4** In this chapter we present two model architectures: the *Base Model* as the state-of-the-art baseline for further experiments, and the novel *Synthetic Boosted Model*, which uses synthetic data to enrich language information. Both architectures are described in detail.

**Chapter 5** This chapter presents the model and the system optimizations. The aim is to find the model parameters which minimize the objective function. To do so, we introduce not only the optimization method and its parameters, but also the series of system adjustments.

**Chapter 6** This chapter presents the experiments results and conclusions. We adapt the *Base Model* architecture to the task conditions, and then the *Synthetic Boosted Model* is explored.

**Chapter 7** In this chapter we do the evaluation on the hold-out dataset, exclusively for the previously selected models.

**Chapter 8** This chapter serves as a summary of our work as well as future directions.

# Chapter 2

# Background

In this chapter, we provide the fundamental knowledge required to understand the key concepts presented in our work. At the very beginning, we strict formulate the problem definition. Afterward, we present the process of transformation an audio sample to spectral input features, which next are used as the input data to the *Deep Neural Network* model. Then in the section 2.3, we introduce the key components of *Deep Neural Network*, where we focus on *Recurrent Neural Networks* (RNN), and optimization techniques commonly used for RNNs. At the end of the chapter, we present Connectionist Temporal Classification algorithm, which allows the RNN models to be successfully used in automatic speech recognition systems.

## 2.1 Problem Definition

Let a single utterance $x^{(i)}$ and transcript $y^{(i)}$ be sampled from a training set defined as $X = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ \dots\}$. Each utterance $x^{(i)}$ is a time-series of length $T^{(i)}$, where every time-slice at $t$ is a vector of spectral input features $x_t^{(i)}$. Each transcript $y^{(i)}$ is a sequence of characters present in the alphabet.

The goal of the model is to convert an input sequence $x^{(i)}$ into an output sequence $y^{(i)}$. At each time-step $t$, the model returns probability distribution over each symbol in the alphabet $\mathbb{P}(\ell|x_t^{(i)})$. In Polish we define an alphabet as $\ell \in \{a, b, c \dots, space, blank\}$[1], where we have added *space* to denote word boundaries and *blank* token due to the objective constrain.

The model output, a sequence of a probability distribution over each symbol in the alphabet, can be decoded to a prediction $\hat{y}$ thanks to the *CTC Inference* algorithm. The

[1] The used polish alphabet is composed of: *a*, *ą*, *b*, *c*, *ć*, *d*, *e*, *ę*, *f*, *g*, *h*, *i*, *j*, *k*, *l*, *ł*, *m*, *n*, *ń*, *o*, *ó*, *p*, *r*, *s*, *ś*, *t*, *u*, *w*, *y*, *z*, *ź*, *ż*, *v*, *x*, *space*, *blank*.

model is trained using the *CTC Loss* function, which measures prediction error. Then we can backpropagate a gradient, update the parameters, and improve model performance. Both the *CTC Inference* and the *CTC Loss* are presented in details in the section Connectionist Temporal Classification.

## 2.2 Spectral Input Features

In this section, we present how a *continuous* speech signal is transformed to the *spectral input features*. Therefore, we introduce the fundamental definitions in signal processing, such as a signal representation in time and frequency domain, needed to understand our feature extraction process. We start from defining the speech as a *discrete* signal in the time domain. For this purpose, we present two basic methods the *sampling* and the *quantization*. Then, we transform a signal into the frequency domain and focus on the *spectrogram* as the short-time frequency analysis tool. Based on a spectrogram, we define our spectral input feature representations, which are *Mel-scale Log Filter Banks*.

The *Mel-scale Log Filter Banks* are the collection of filters scaled according to the *mel-scale* [1]. The reason of use filter banks as input features is that the cochlea, in the human ear, resembles a filter bank. Additionally, the filter banks ranges are scaled using Mel-Scale which reflects humans frequency perception. Both the filter bank and the mel-scale tries to mirror humans speech perception.

### Signal Representation in Time Domain

Humans make speech sounds using speech organs such as the tongue, lips, and palate. The result of this complex process is the *sound pressure* change, which is a variation of air pressure caused by a sound wave. The amount of sound pressure change is measured by *amplitude*. A speech waveform is a representation of sound in the time domain, which shows changes of amplitude through time. Speech is an analog signal, where the range and the domain of a signal are continuous. Analog signals are hard to be processed on computers, so they need to be converted into digital signals, of which domains and ranges are discrete.

To achieve a discrete domain, we have to measure the signal's value at specific points of interest. This process is called as the *uniform sampling*. Let $x_a(t)$ be a continuous signal as a function of time $t$. If we sample $x_a$ with a sampling period $T$, the output digital signal of this process is $x[n] = x_a(nT)$. The *sampling frequency* $F_s$ is defined as the inverse of the sampling period $F_s = 1/T$, where its unit is hertz (Hz).

To achieve a discrete range, the continuous values of the signal have to be converted

into a discrete set of values. This process is called as the *quantization*. In an audio signal, the quantization level is defined as *bits depth*, which represents the number of bits needed to represent the range of the signal. For example, values of the 16-bit signal are in the range from -32768 to 32767.

Both processes the sampling and the quantization can cause the loss in signal information (additional noises). The *sampling frequency* and the *bit depth* need to be high enough in order to achieve a good approximation of the original continuous signal. The main difficulty is to choose the right trade-off between the discrete signal quality and its size.

### Pre-emphasis Audio Signal

At the very beginning, we apply the pre-emphasis filter on the raw discrete audio signal to amplify the high frequencies. The pre-emphasis is not a critical step, but it is useful to balance the frequency spectrum since high frequencies usually have smaller magnitudes compared to lower frequencies. The pre-emphasis filter is applied to a speech signal $x[n]$ as:

$$y[n] = x[n] - \alpha \cdot x[n-1] \tag{2.1}$$

where $\alpha$ is common in the range from 0.95 to 0.97.

### Signal Representation in Frequency Domain

The *Frequency domain* is a different perspective to look at a signal besides the time domain. Speech signal can be represented as a composition of the *pure* tones, which correspond specific frequencies. The process of signal decomposition, changing a signal from the time domain to the frequency domain, is called as the *spectral transformation*.

Speech signals are non-stationary, which means that the statistical properties of the signal change over time (intensity, variance, ...). The speech signal can be considered as stationary and periodic in short time intervals. Therefore we are able to analyze the *short-time* intervals using the *Fourier transformation*. The short-time constraints lead us to a set of techniques called the *Short-Time Analysis*. The idea is to split a signal into short time frames (usually overlapped), wherein we consider a signal in one frame as stationary, and then transform each frame separately. However, we have to be aware that frames should be short enough to satisfy assumptions, in practice, 10 to 40 ms.

Given the speech signal $x[n]$, the *short-time signal* $x_m[n]$ of the frame $m$ is defined as:

$$x_m[n] = x[n] \cdot w_m[n] \tag{2.2}$$

with $w_m[n]$ is the window function. The window function is the same for all frames, so it can be simplified as:

$$w_m[n] = w[m-n] \tag{2.3}$$

with $0 \leq n \leq N-1$, where $N$ is the frame length. We assured the audio to be periodic by framing the signal. Additionally, we apply the window function $w[n]$ on every frame to avoid artifacts on window edges, and to reduce the signal discontinuity. Therefore we can use the *Hamming* function [3]:

$$w[n] = 0.54 - 0.46 \cdot \cos(\frac{2\pi n}{N-1}) \tag{2.4}$$

Each frame $m$ of a signal is Fourier transformed, and the results are collected in the matrix, which gathers the magnitude (a phase is usually neglected) for each point in time and frequency. This can be expressed as:

$$X(m,\omega) = \sum_{n=-\infty}^{\infty} x[n]w[n-m]e^{-j\omega n} \tag{2.5}$$

Then the spectrogram magnitude is computed as:

$$S(m,\omega) = |X(m,\omega)^2| \tag{2.6}$$

The spectrograms can be the *wide-band* and the *narrow-band* depend on the window length. Wide-band spectrograms use a short window length (<10 ms) which corresponds to filters with a wide bandwidth (>200 Hz). In opposite, narrow-band spectrograms use a longer window (>20 ms) which leads to narrow bandwidth (<100 Hz). In consequence, two kinds of spectrograms differ in time and frequency representation. Wide-band spectrograms have a good *time resolution*, so they are able to track pitches precisely. In contrast, narrow-band spectrograms give a good view of the *frequency resolution*. Nevertheless, rapid changes over time are harder to distinguished.

## Mel-scale Log Filter Banks

The filter-banks is the collection of filters, which are spread out over audible frequencies, and measure the energy amount within the part of the signal spectrum. The mel spaced filter-banks are an attempt to reflect closely the human perception [1]. The mel is a unit of "measure of perceived pitch or frequency of a tone". In 1940, Stevens and Volkman assigned 1000 mels as 1000 Hz, and asked participants to change the frequency until they perceived the pitch changed some proportions with regard to the reference. The threshold frequencies were marked, resulting in the mapping between the real frequency scale (in

Hz) and the perceived frequency scale (in mel). The popular formula to convert from the frequency scale to the mel scale is:

$$f_{mel} = 1125 \ln(1 + \frac{f_{Hz}}{700}). \tag{2.7}$$

where $f_{mel}$ is the frequency in mels and $f_{Hz}$ is the frequency in Hz. According to the mel scale, humans are more sensitive to lower frequencies (figure 2.1 on the left).

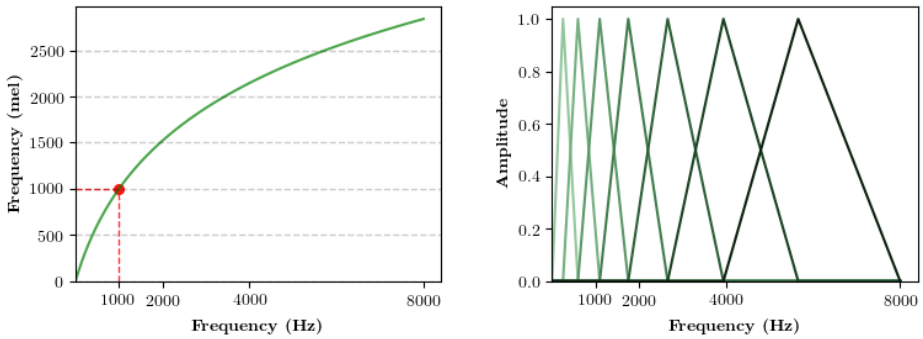


Figure 2.1: The relationship between the frequency scale and mel scale is shown on the left. The Mel-scale filter-bank composed of 7 filters is presented on the right.

The mel-scale filter-banks contains $M$ filters, where the number of filters defines the output resolution. Each filter has a triangular shape and is spaced uniformly on the mel scale. The filter functions are defined as:

$$H_m(k) = \begin{cases} 0 & k < f(m-1) \\ \dfrac{k - f(m-1)}{f(m) - f(m-1)} & f(m-1) \leq k < f(m) \\ 1 & k = f(m) \\ \dfrac{f(m+1) - k}{f(m+1) - f(m)} & f(m) < k \leq f(m+1) \\ 0 & k > f(m+1) \end{cases} \tag{2.8}$$

where $H_m$ denotes the magnitude of the filter $m$, the frequency is denoted by $k$ and the vector $f$ contains $M + 2$ linearly spaced filter border values (figure 2.1 on the right). To sum up, the mel-filter banks is composed of the high filter resolution where human hearing is precise and the low resolution where it is inaccurate. At the end of the feature extraction process, we use the logarithm to enhance signal details, which leads to the *Mel-scale Log Filter Banks* representation.

## 2.3 Deep Neural Networks

Our models are entirely based on the *Deep Neural Networks* (DNN), which are used for the function approximation. The DNN is a stack of hidden layers, computational nodes which are differentiable. Our models are composed of many standard building blocks like the linear layer, the convolutional layer, and several nonlinear function such as the logistic sigmoid function (sigmoid), the hyperbolic tangent function (tanh) or the rectified nonlinear unit (relu). In this section we briefly introduce the *Recurrent Neural Networks* (RNN) and optimization techniques which are common for the RNN. Other more novel or sophisticated building blocks such as the LSTM are introduced in the section where they are used.

### Recurrent Neural Networks

To use an arbitrary amount of temporal context, what is crucial in speech recognition, we use the *Recurrent Neural Networks* (RNN). The RNN build the hidden representation $h_t$ in time $t$ based on the current input $x_t$, and the previous hidden state $h_{t-1}$:

$$h_t = f(x_t, h_{t-1}). \tag{2.9}$$

A vanilla RNN model $f$ is defined as:

$$h_t = g(Wx_t + Uh_{t-1} + b) \tag{2.10}$$

where $g$ is a nonlinearity function such as the rectified nonlinear unit (relu). The weights $W$, $U$, and $b$ are trainable parameters. Moreover, the RNN can be the *bidirectional*, where past $h_{t-1}$ and future $h_{t+1}$ information is taken into account in building representation $h_t$. The equations for the bidirectional RNN are:

$$\overrightarrow{h}_t = f(x_t, \overrightarrow{h}_{t-1}) \tag{2.11}$$

$$\overleftarrow{h}_t = f(x_t, \overleftarrow{h}_{t-1}) \tag{2.12}$$

$$h_t = \overrightarrow{h}_t + \overleftarrow{h}_t \tag{2.13}$$

In this case, we sum the forward and backward hidden states, but also a common practice is to concatenate them.

### Optimization

In our work, we use a variation of the *Stochastic Gradient Descent* (SGD) to optimize the model's parameters. The method is called the stochastic because it uses randomly

selected samples to evaluate the gradients, hence the SGD can be regarded as a stochastic approximation of gradient descent optimization. The basic SGD update is given by:

$$\theta_t = \theta_{t-1} - \eta_t \nabla_\theta \mathcal{L}(X, Y) \tag{2.14}$$

where $\mathcal{L}(X, Y)$ is the loss function computed on the training pair $(X, Y)$ and $\eta_t$ is the *learning rate* used at time $t$. The *mini-batch* is commonly used due to computationally constrains. Moreover, it can reduce fluctuations, as the gradient computed at each step is averaged over more training examples.

The *momentum* augments the plain SGD algorithm with a velocity term and can accelerate learning (an analogy to momentum in physics):

$$v_t = \rho v_{t-1} + \eta_t \nabla_\theta \mathcal{L}(X, Y) \tag{2.15}$$

$$\theta_t = \theta_{t-1} - v_t \tag{2.16}$$

Here the parameter $\rho$ dictates the amount of momentum to use and is typically in the range $[0.9, 0.99]$. Unlike in classical stochastic gradient descent, it tends to keep traveling in the same direction, preventing oscillations.

There are vast alternative methods which attempt to blend estimations of the first or second-order information of the gradient. They try to approximate the *Hessian* information, which is in practice hard to achieve. In our work, we use the *Adaptive Moment Estimation* [26], which running averages of both the gradients, and the second moments of the gradients. Adam's parameter update is given by:

$$m_w^{(t+1)} \leftarrow \beta_1 m_w^{(t)} + (1 - \beta_1) \nabla_w L^{(t)} \tag{2.17}$$

$$v_w^{(t+1)} \leftarrow \beta_2 v_w^{(t)} + (1 - \beta_2) (\nabla_w L^{(t)})^2 \tag{2.18}$$

$$\hat{m}_w = \frac{m_w^{(t+1)}}{1 - (\beta_1)^{t+1}} \tag{2.19}$$

$$\hat{v}_w = \frac{v_w^{(t+1)}}{1 - (\beta_2)^{t+1}} \tag{2.20}$$

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \frac{\hat{m}_w}{\sqrt{\hat{v}_w} + \epsilon} \tag{2.21}$$

where $\epsilon$ is a small scalar used to prevent division by 0, and $\beta_1$ and $\beta_2$ are the forgetting factors for gradients and second moments of gradients, respectively. Squaring and square-rooting are done elementwise. With $(\beta_1)^{t+1}$ and $(\beta_2)^{t+1}$ we denote $\beta_1$ and $\beta_2$ to the power $t + 1$.

## 2.4 Connectionist Temporal Classification

The dataset contains pairs of spectral input features $x^{(i)}$ and corresponding transcript $y^{(i)}$, but we do not know how transcript aligns to the input. To hand-align each character to its location is indeed time-consuming. The attempts of hand-drafted rule-based approaches fail, because both length and ratio of length vary between each sequence $x^{(i)}$ and $y^{(i)}$. In consequence, we can not directly use the supervised classification algorithms. The *Connectionist Temporal Classification* (CTC) is the way to overcome the lack of alignment between the input and the output [10]. The model for a sequence $x^{(i)}$ returns a sequence of distribution over all possible elements, which can appear in $y^{(i)}$. The CTC algorithm use this distributions either to assess the *probability* of a correct output (*Loss Function*) or to *infer* a most probable output (*Inference*). In both cases, they use the key concept of the *Alignment*, which we briefly presented at the beginning of this section. Then we introduce the *Loss Function* and the *Inference* algorithms as the crucial parts of each CTC based ASR system. We encourage to check the CTC in-depth analysis presented in [42], especially to distinguish a connection to other architectures such as *encoder-decoder*.

### 2.4.1 Alignment

The sequence of allowed elements, which can appear in $y^{(i)}$, forms the *alignment*. The CTC appends the special *blank* token to the alphabet. Using the *blank* token, we address two issues. Firstly, it does not make sense to force a model to predict a character when nothing is there. Secondly, it enables to predict repeated characters in a row.

The model returns the vast number of alignments, but only some of them are valid. The valid alignments is a set of alignments which collapse (*marginalize*) to the same output. The CTC marginalizes alignments in two steps. It merges repeated elements and then removes *blank* tokens. Finally, summing the probability of valid alignments, the CTC can estimate the output probability, which is used either to calculate the *loss* or to do *inference*.

Worth to sum up the fundamental properties of CTC alignments. The alignments between $x^{(i)}$ and $y^{(i)}$ are monotonic and in a relation many-to-one, what implies a third property that the length of $y^{(i)}$ cannot be greater than the length of $x^{(i)}$.

### 2.4.2 Loss Function

The *CTC Loss Function* $\mathcal{L}$ for a single $(X, Y)$ pair is defined as:

$$\mathbb{P}(Y \mid X) = \sum_{A \in \mathcal{A}_{X,Y}} \prod_{t=1}^{T} p_t(a_t \mid X) \tag{2.22}$$

$$\mathcal{L}_{CTC}(X, Y) = -\log \mathbb{P}(Y \mid X) \tag{2.23}$$

where $\mathcal{A}_{X,Y}$ describes valid alignments. The model's parameters are tuned to minimize the negative log-likelihood. The CTC conditional probability marginalizes over the set of valid alignments computing the probability for a single alignment step-by-step. The straightforward approach scoring each alignment and sum them up fails due to the massive number of alignments. Thanks to the dynamic programming algorithm, we can efficiently compute the loss function. The crucial insight is that two alignments can be merged if they have reached the same output at the same time [42].

The *CTC Loss* is fully differentiable with respect to the prediction at each time step, since the objective just sums the product of them. In result, it is possible to compute the gradient, and run the backpropagation as usual.

### 2.4.3 Inference

After the model training, we want to find the most probable output $\hat{Y}$ for a given input $X$, what can be described as:

$$\hat{Y} = \underset{Y}{\operatorname{argmax}}\, p(Y \mid X) \tag{2.24}$$

To have in mind fact that a single output can have many alignments, we have to use the beam search. A regular beam search at each time step generates the new set of hypothesis, which extends the previous hypothesis with all possible characters, and then it keeps only the most probable candidates. The beam search can be modified to store the output prefixes solely after a marginalize process [42].

# Chapter 3

# Data

The data set $X = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots\}$ consists of audio samples, sequences of spectral input features $x^{(i)}$, and the corresponding transcript composed of the character sequence $y^{(i)}$ (Chapter Background). The vast dataset is crucial to achieving the exceptional performance of speech recognition systems, especially in the case of the systems based on a deep neural network. The model tries to identify patterns presented in a dataset. Therefore the corpus must be of high quality because the model can easily adapt to anomalies or particular characteristics of the training dataset. We analyze the corpus with particular care before starting to work with our models. The data defines which model architecture is the most convenient.

## 3.1 Audio Representation

Audio samples have the single-channel (mono) with the sampling rate of 16 kHz and the depth of 16 bits. Input data $x^{(i)}$ to the model is the audio signal presented in the form of the *Mel-scale log filter banks* (however also the use of the raw speech is presented in [56]). In this representation, the energy in the individual bands of the spectrogram is scaled using the *mel scale*. The use of more sophisticated representations such as *Mel Frequency Cepstral Coefficients* (*MFCC*) [16] does not yield noticeable improvements, so we deem additional transformations as needless. The table 3.1 shows the *Mel-scale log filter banks* transformation parameters, which gives the best results for the Polish language.

The dataset is normalized so the energy level for each sample remains constant. For each *filter bank* in the *train* dataset, we calculate the global average and standard deviation. Then base on these statistics, we normalized the *train*, the *dev* and the *test* datasets. Alternatively, one can use the *Batch Normalization* as the first model layer to achieve a similar behaviour [33]. However, a large dataset is often highly inconsistent,

| Pre-emphasis | Winlen (ms) | Winstep (ms) | Winfunc | NFFT | Filterbanks |
|---|---|---|---|---|---|
| 0.95 | 25 | 10 | hamming | 512 | 80 |

Table 3.1: The transformation parameters: the pre-emphasis coefficient, the window length, the step size, the window function, the DFT size, the number of filterbank energies.

so the statistics $(\mu, \sigma)$ vary considerably for different batches. In consequence, the model is subjected to the strong regularization, which in our experiments negatively affects the model's performance.

## 3.2 Dataset Construction

The vast majority of transcribed audio datasets are closed, and those concerning the Polish language in particular. One of the few publicly available datasets is the *Clarin-PL* [43], created by the European initiative the *Common Language Resources and Technology Infrastructure* (Clarin) [12]. The polish fragment *Clarin-PL* is small (36 hours), but in our work we use additionally the closed dataset *Jurisdic* [11] which is over 10 times larger than the *Clarin-PL*. Both the *Clarin-PL* and the *Jurisdic* are well documented, and contain a variety of recording sources, including the *read speech*, the *spontaneous dictation*, and the *phone calls*.

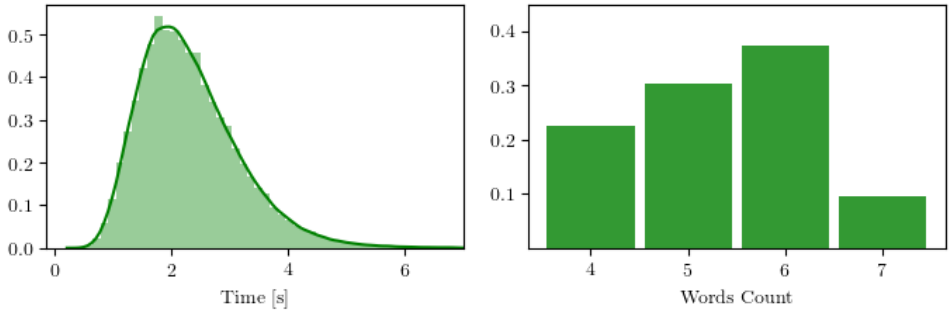


Figure 3.1: Histograms of the samples from the training dataset aggregated according to: the length of samples (left) and the number of words (right).

The data comes from a variety of sources is highly non-uniform, both in terms of audio data and transcriptions. The number of words in recordings ranges from one to several dozen. Therefore, we perform the segmentation of samples into single words on both corpora, and then combine into utterances from 4 to 7 words. In consequence, the model is not able to recognize longer dependencies (more than 7 words), which is a reasonable

assumption in the context of this work. However, the samples are more homogeneous, and the number of observations is increased significantly. The figure 3.1 shows the distribution of samples by the length of recordings, and the number of words in the transcription after segmentation.

| Corpus | Size (hours) | Samples (k) |
|---|---|---|
| Clarin-PL (train) | 34 | 41 |
| Jurisdic | 351 | 578 |

Table 3.2: The components of the training dataset. The number of samples contains from 3 to 7 words (after the segmentation).

The dataset *Clarin-PL* is divided into 3 parts the *train*, the *dev*, and the *test*. The proportions of the division are as follows 80%, 10%, and 10%. Each part has independent speakers and unique transcriptions. The data from the corpus *Jurisdic* do not appear in the *dev* and the *test* datasets, because most transcriptions (82%) are duplicated (details how it was built can be found in [11]). The figure 3.2 shows the distribution of the duplicated transcriptions in the training dataset. Speech recognition systems *end-to-end* learn to recognize *phonemes* parallel to learning the language model. In this case, the language model has a strong *bias* towards repetitive words. The knowledge included in the much larger *Jurisdic* corpus is important, so it is entirely included into the training dataset (table 3.2).

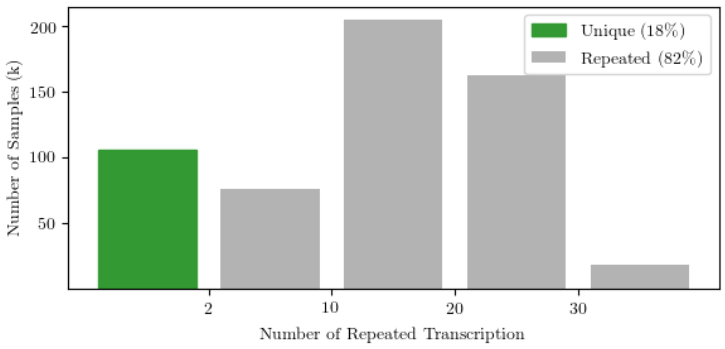


Figure 3.2: The distribution of the number of transcription repetitions in the training dataset. Samples with the unique transcript are marked in green. The grey color represents the samples in which transcriptions are repeated.

### Synthetic Dataset

In this work, we present a model that derives language knowledge from synthetic data. For this purpose, we generate the auxiliary synthetic corpus of over ten thousand hours.[1] Due to the high accessibility, we choose a commercial *Text To Speech* (*TTS*) model. The system based on the *WaveNet* [39] generated files based on transcripts created from the publicly available text corpus *PolEval* [48]. All transcriptions are composed of 7 consecutive correct tokens, not including punctuation marks.[2] The created synthetic speech corpus is publicly available.

## 3.3 Data Augmentation

The data augmentation is a strategy that enables to increase the diversity of data available for training significantly. Most of the augmentation methods in speech recognition systems operate directly on audio data [19, 27, 34]. The different approach *Spec Augment* [55] is the augmentation method which operates on the transformed features, for instance, the spectrogram bands. The method is simple and based on three stages: wrapping the features, masking blocks of frequency channels, and masking blocks of time steps. In this work, we use the last two parts that have a definite impact on the results and can be applied with a low computing cost. The sample masking makes the model more resistant to distortion and fading of information both in the time and frequency domain. The figure 3.3 shows an example of augmentation.

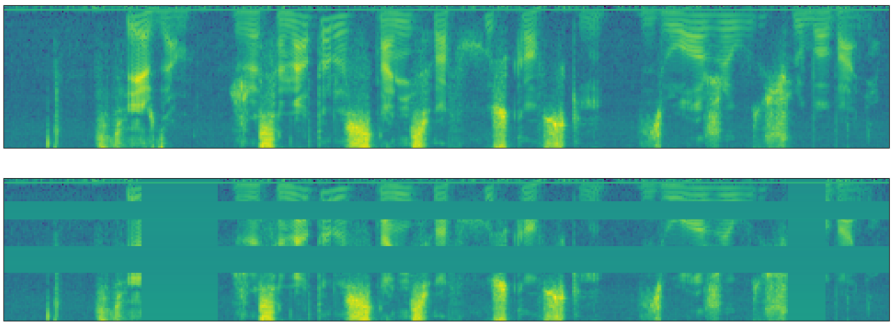

Figure 3.3: The example of augmentation of a 500-frame sample: the original sample (top) and the sample after augmentation (bottom). The parameters are respectively $m_f = 2, F = 20, m_t = 2, T = 50$.

[1] Unfortunately, due to the limited duration of the research, we have only used a part of the synthetic corpus.

[2] A token is a string of contiguous characters between two spaces, or between a space and punctuation marks. The correct token means that it is constructed only from chars available in the model alphabet $\ell$.

High values of $T$ parameter shown in [55], in the range from 50 to 100 frames, mask a significant part of the word. The augmentation method, masking long fragments of the audio, forces the model to make a prediction based on the context of other words. The technique of masking a fragment of input data is strongly associated with language modeling strategies [20, 45, 49, 51]. Unfortunately, in the case of inflected languages (including Polish), the proposed technique in the time domain does not improve the performance. Large masks (e.g. $T = 100$) lead to the training instability, while smaller ones (e.g. $T = 20$) bring small benefits.

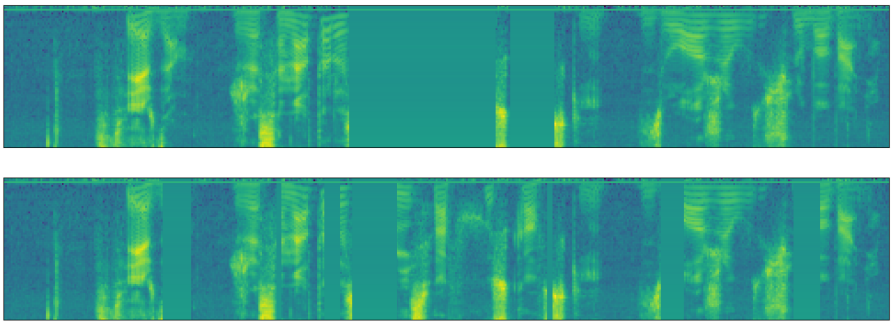

Figure 3.4: The original augmentation method (top) with parameters $m_t = 2, T = 100$ and modified (down) with parameters $T_{ratio} = 0.1$, $T =< 5.25 >$, $T = 10$. The modified method is supposed to mask single phonemes rather than words, and prevent to overlap masks.

We adjust the data augmentation technique, in the time domain, to the specificity of the inflected language, which requires more attention to words endings (figure 3.4). We have observed that we achieve better results when we use a few small masks. In our case, the mask should try to cover single phonemes, not whole words. The proposed augmentation technique is controlled by three parameters: $T$, $T_{ratio}$ and $T_{space}$. The $T$ parameter specifies the range from which the mask width is drawn (*uniform distribution*). The best results we obtain for the range $T =< 5, 15 >$, which is correlated with the statistical length of phonemes in Polish [18]. The $T_{ratio}$ parameter describes the ratio of masked parts to the entire sample. The $T_{space}$ parameter specifies the minimum gap between masked sections. Close (or overlapped) masks lead in our case to the training disruptions, because the model is not able to handle large mask at all.

# Chapter 4

# Model

The goal of the model is to transform a sequence of spectral input features $x^{(i)}$ into a transcript sequence $y^{(i)}$. In fact, the model for each time step $t$ in sequence $x^{(i)}$ estimates the probability distributions $\mathbb{P}(\ell|x_t^{(i)})$ over each letter $\ell$ in the alphabet. Then, the sequence of probability distributions is decoded using the *CTC Inference* algorithm, which returns the most probable transcript $\hat{y}^{(i)}$. Additionally, one can extend a decoding process and use the *External Language Model* either the char or the word-based, which helps to correct minor language mistakes.

The model solves two closely related tasks. It learns to recognize phonemes and formulate grammar rules at the same time. Conceptually, the model can be divided into two sub-models *Phoneme Model*, and *Implicit Language Model*, where a boundary between them is fuzzy. In fact, a model is able to parallel and accurate build both of them, when a training corpus is large enough. However, inflected languages such as Polish contains much more grammar rules to define than in the case of English. In Polish, there are complex grammar rules such as ('ó/u', 'ż/rz' or 't/d'), silent letters, and numerous prefixes and suffixes. Therefore, in order to achieve comparable results in the Polish language, the corpus must be substantially larger than the one presented for the English language [30].

In our case, the Polish language is complex, the training corpus is small and the transcriptions are duplicated (Chapter Data), that's why the *Implicit Language Model* is imprecise. The inflection of the Polish language and the imprecise model yield the mistakes with a large *char edit distance*, that are hard to decode, in particular using external static language models (either the char or word-based *n-gram models*). A dedicated, separate, and more sophisticated external language model can be created, and as a decoder can try to correct mistakes. However, still, the decoder can not solve complex problems, because it operates on sparse probability distributions returned by the model.

In this chapter we present two models the *Base Model* and the *Synthetic Boosted Model*. The base model is a recurrent neural network with the objective *CTC Loss* (Chapter Background). Whereas the *Synthetic Boosted Model* is an extension of the base model with the built-in decoder *Synthetic Language Model*, which, thanks to the synthetic data, increases the amount of language information supplied to the model. The *Synthetic Language Model* is able to solve complex problems because it operates on hidden states $h$ of the base model, which are rich in information.

## 4.1 Base Model

The base model architecture is built upon the *Deep Speech 2* model, which is the continuation of the *Deep Speech* success [25, 30]. The model contains a convolutional layer and then several recurrent layers (figure 4.1). At the end of the model, there is an output layer that, for each time step $t$ in the input sequence $x^{(i)}$, returns the probability distributions $\mathbb{P}(\ell|x_t^{(i)})$ over each letter $\ell$ in the alphabet.

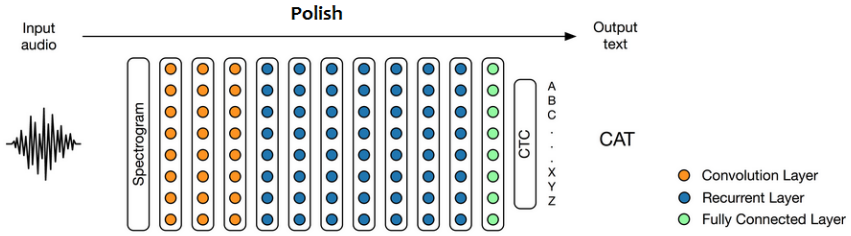


Figure 4.1: The architecture of the Base Model. The figure is adopted from [40].

### Convolutional Layer

The first layer of the *Base Model* is a two-dimensional (*time-and-frequency domain*) convolutional layer. The layer analyses a small context bounded by a *receptive field* with size in frequency $c_f$ and time $c_t$. The layer representation is defined by $h^l$, assuming that $h^0$ represents input data $x$. Convolutional layer activations can be written for the $i$-th activation at time $t$ and frequency $f$ as:

$$h^l_{t,f,i} = \mathcal{F}(w^l_i * h^{l-1}_{t-ct,t+ct:f-cf,f+cf}) \tag{4.1}$$

where $*$ means a convolve of the $i$-th filter with the receptive field of the input data, and $\mathcal{F}(\cdot)$ is an unary, non-linear function. We have selected the *Clipped Rectified-linear* (ReLU) $\sigma(x) = \min\{\max\{x, 0\}, 20\}$, as an activation function. We used the *same* convo-

lution to keep constant the number of features in both frequency and time.[1]

For a long time, ASR systems used *Temporal Convolution*, which is a one-dimensional convolutional layer (along the time dimension). It analyzes the small context of a signal, which is composed of concatenation of previous and subsequent $C$ input feature vectors [13, 17, 25]. However, the two-dimensional convolutional layer more precisely reflects the audio *spectral variance* by sharing filter weights also in the frequency domain. In consequence, the two-dimensional convolutional layer reduces the number of parameters, is resistant to shifts in frequency ranges, and significantly accelerates optimization, when the *stride* is applied. Therefore the two-dimensional convolutional layer is often used in ASR systems [15, 21, 28].

## Recurrent Layers

The next layers of the *Base Model* are the bidirectional recurrent layers, which constitute the core of the model, because they contain more than 95% of network parameters. Either the *fully connected* layer or the *convolutional* layer builds the $h_t^l$ representation exclusively based on the preceding representation $h_t^{l-1}$. The recurrent layer is unique because it also has the loop $h_{t-1}$ which in fact gives a view to all previous steps. In consequence, a representation $h_t^l$ is built upon the broad backward context.

Moreover, the bidirectional recurrent layer exists, which has at time $t$ both the backward context $h_{t-1}$, and the forward context $h_{t+1}$. The forward $\overrightarrow{h}^l$ and backward $\overleftarrow{h}^l$ layer activations at time $t$ are defined as:

$$\overrightarrow{h}_t^l = \mathcal{F}(h_t^{l-1}, \overrightarrow{h}_{t-1}^l) \tag{4.2}$$

$$\overleftarrow{h}_t^l = \mathcal{F}(h_t^{l-1}, \overleftarrow{h}_{t+1}^l) \tag{4.3}$$

Then the components are summed, and the bidirectional recurrent layer activation is equal to $h_t^l = \overrightarrow{h}_t^l + \overleftarrow{h}_t^l$. There are many different variants of the function $\mathcal{F}(\cdot)$, which works successfully in the ASR. In the base model, we selected the widely used *Long Short-Term Memory* (LSTM) layer [6] (and explained in [36]), which can discover long dependencies and has publicly available highly efficient implementation.

The LSTM layer, in relation to a vanilla RNN presented in the Chapter Background, has in addition the cell state, the vector $C_t$, which describes the *general* context at time $t$. The cell state is modified by two gates the *Forget gate layer* $f_t$ and the *Input gate layer*

[1] The activation function is limited to avoid exploding of the gradient (in particular the first phase of training).

$i_t$ based on vectors $h_t^{l-1}$ and $h_{t-1}^l$:

$$f_t = \sigma(\mathbb{W}^{(f)} \cdot [h_t^{l-1}, h_{t-1}^l] + b^{(f)}) \tag{4.4}$$

$$i_t = \sigma(\mathbb{W}^{(i)} \cdot [h_t^{l-1}, h_{t-1}^l] + b^{(i)}) \tag{4.5}$$

$$\tilde{C}_t = \tanh(\mathbb{W}^{(C)} \cdot [h_t^{l-1}, h_{t-1}^l] + b^{(C)}) \tag{4.6}$$

$$C_t = f_t \circ C_{t-1} + t_t \circ \tilde{C}_t \tag{4.7}$$

where $\circ$ means a pointwise multiplication. The *Forget Gate Layer* decides which features to erase in the general context $C_{t-1}$. The *tanh* gate creates the group of potential candidates for new features $\tilde{C}_t$, and the *Input Gate Layer* defines how important they are in the general context $C_t$. At the end, there is the *Output Gate Layer*, which returns the representation $h_t^l$ by the view of the current cell state $C_t$ via the filter $o_t$:

$$o_t = \sigma(\mathbb{W}^{(o)} \cdot [h_t^{l-1}, h_{t-1}^l] + b^{(o)}) \tag{4.8}$$

$$h_t = o_t \circ \tanh C_t \tag{4.9}$$

The recurrent layers are the essential elements of the model. Thanks to the recurrent layers, a model based on the large training dataset, can build the rich *Implicit Language Model*.

### Output Layer

At the end of the model, there is a linear layer with the activation function *softmax*, which returns, for each time step $t$ in the input sequence $x^{(i)}$, the probability distribution of occurrence of each character from the alphabet $\ell$ at time $t$:

$$\mathbb{P}(\ell_t = k|x) = \frac{\exp(w_k^L \cdot h_t^{L-1})}{\sum_j \exp(w_j^L \cdot h_t^{L-1})} \tag{4.10}$$

## 4.2 Synthetic Boosted Model

The *Synthetic Boosted Model* is composed of two integrated parts, the *Phoneme Model* and the *Synthetic Language Model*, which are optimized in the separated trainings (figure 4.2). The *Synthetic Boosted Model* formulates the grammar rules using a much larger, synthetically augmented training corpus. The model for the input sequence $x^{(i)}$ returns, as the base model does, the sequence of probability distributions $\mathbb{P}(\ell|x_t^{(i)})$ over each letter $\ell$ in the alphabet for each time step $t$.

The use of a synthetic data up to the ration 1 to 1, an authentic to a synthetic data, is a widely practiced technique of augmentation the training dataset. Naively adding more synthetic data to a training dataset brings a decline in performance. The authentic

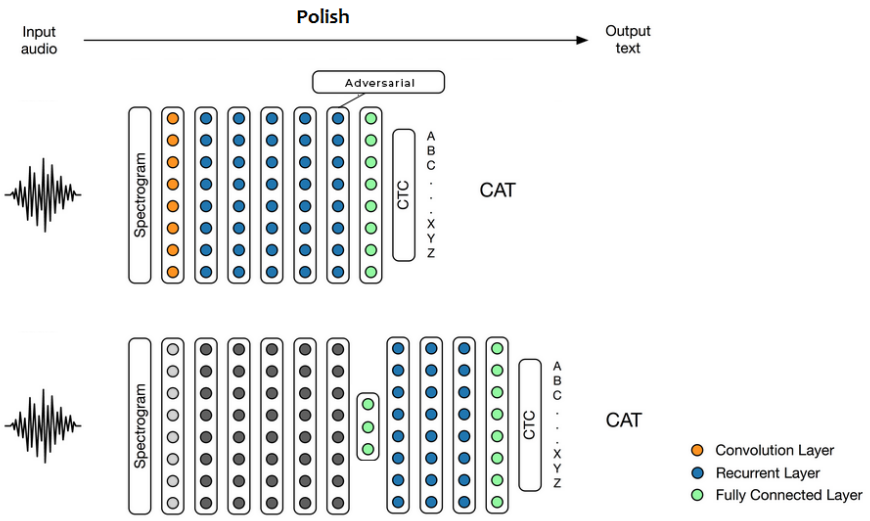


Figure 4.2: The architecture of the Synthetic Boosted Model. The figure is adopted from [40].

samples are overwhelmed by the synthetic ones, what in consequence, reduces the ability of actual phoneme recognition. Therefore, training is divided into two stages. Firstly, we train the *Phoneme Model* (upper figure 4.2) on a dataset with the ratio 1 to 1, an authentic to a synthetic data. Then, we train the *Synthetic Language Model* (lower figure 4.2), which processes several times larger synthetically augmented training corpus. The *Phoneme Model* is frozen (weights are not updated) to protect the ability of actual phoneme recognition, therefore it plays a feature extractor role exclusively at this stage. Finally, the purpose of the *Synthetic Language Model* is to *boost* the representation $h$ created by the frozen *Phoneme Model*, and return the correct prediction.

## Phoneme Model

The *Phoneme Model*, like the base model, is made up of the convolutional layer, and then from several recurrent layers. It also has an output layer, which is removed after training. The model is optimized with the ratio 1 to 1, an authentic to a synthetic data, with the modified loss function.

The aim is to build the accurate *Phoneme Model*, which also produces the independent of data source representations $h$, the last LSTM hidden states. For easily distinguishable representations, the *Synthetic Language Model* constructs separate rules for authentic and synthetic data, which is an undesirable phenomenon. To achieve more independent representations, we add the *adversarial* component to the loss function. We propose the simple function that measures in every *mini-batch* the average activation vectors for both

authentic $\bar{\mathbf{h}}_a$ and synthetic samples $\bar{\mathbf{h}}_s$, and then calculates their *Manhattan* distance:

$$\mathcal{L}_{Adv} = \|\bar{\mathbf{h}}_a - \bar{\mathbf{h}}_s\|_1 \tag{4.11}$$

In consequence, the loss function has two components the *CTC Loss* and the *Adversarial Loss*, and is defined as the weighted sum:

$$\mathcal{L} = w_0 \cdot \mathcal{L}_{CTC} + w_1 \cdot \mathcal{L}_{Adv} \tag{4.12}$$

where in every single *mini-batch* we provide an equal number of authentic and synthetic samples.

### Synthetic Language Model

The *Synthetic Language Model* is made up of the narrow *fully-connected* layer (called the *projection*), and then several recurrent layers, and an *output Layer*, which are the same as in the base model. Despite the modified loss function imposed on the *Phoneme Model*, representations $h$ are still highly dependent on the source of data. Therefore, we introduce the *projection* layer, which forces to reduce the number of dimensions. Thanks to dimension reduction, the representations are more independent of the data source, but much of the information is lost.

The *Phoneme Model* seamlessly learns to correctly recognize synthetic data. The aim is to build the training process, where the *Phoneme Model* makes *similar* mistakes on synthetic data as on authentic data. For this purpose, we mask randomly selected parts of the spectrogram (Chapter Data). In this way, we simulate the situation in which the *Phoneme Model* has difficulty to recognize the phoneme on authentic data, and the *Synthetic Language Model* tries to correct it. The process is analogous to an unsupervised language modeling on raw text. Based on the context, the model tries to estimate the masked part of the input. In consequence, we can learn language dependencies processing masked synthetic data.

# Chapter 5

# Optimization

The optimization aims to find the model parameters $\theta$, for which we minimize the loss function $\mathcal{L}(\hat{y}, y)$. For this purpose, we calculate the gradient $\nabla_\theta \mathcal{L}(\hat{y}, y)$, and iteratively update the parameters $\theta$ according to the optimization method, which in our case is the *Adaptive Moment Estimation* (*Adam*) method [26]. The *Adam* method is the continuation of the iterative optimization method *Stochastic Gradient Descent* (*SGD*), which averages both the gradient and the second moment of the gradient (Chapter Background).

The optimization can be conceptually divided into two, closely related parts. In the beginning, we present the process of model optimization. We focus on two basic parameters in particular, the *learning rate* and the *batch size*. Then in the next section, we present the essential system optimizations. To perform a series of experiments in highly limited time is only possible if the optimization process is adapted to the available infrastructure. The efficient implementation of the system is crucial because it allows doing experiments in a reasonable time.

## 5.1 Model Optimization

The optimization method is the *Adam*, and the key parameters of the optimization method are the *batch size* and the *learning rate* (Chapter Background). The forgetting factors for gradients and second moments of gradients, respectively, equals $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

The *CTC Loss* function tends to reach high values for longer utterances. The high values of the loss function destabilize the training process at an early stage of optimization in particular. Therefore for the first epoch, we sort samples by increasing length, as described in [30]. However, in our case, this approach is not essential because samples are trimmed up to 7 words (Chapter Data).

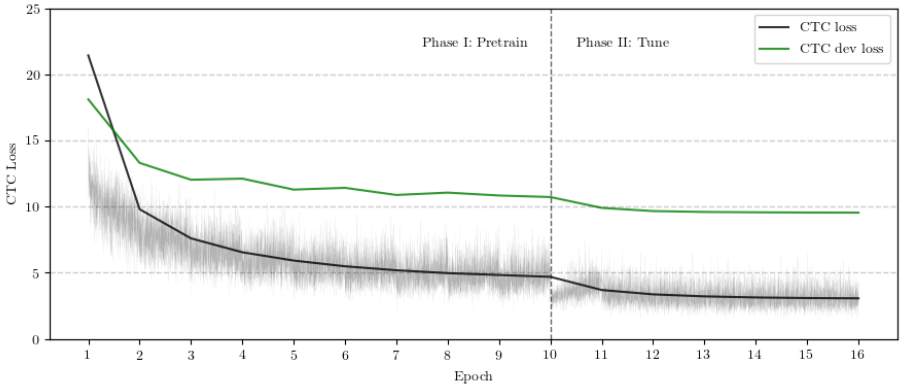


Figure 5.1: The optimization process of the arbitrary model without an augmentation technique. The optimization is divided into two phase: *Pretrain* (left) and *Tune* (right).

## Learning Rate

Although the optimization method *Adam* belongs to the *adaptive gradient algorithm* group of methods, the predefined *learning rate* parameter $\eta$ still has the significant impact on the optimization process. To define a priori the accurate learning rate $\eta$ is a tough task. We do a set of initial experiments to define an appropriate learning rate scheduler, which finally is composed of two phases. In the first *pretrain* phase, the learning rate is constant, and equals $\eta = 10^{-3}$. The relatively large learning rate aims to overcome a high *plateau*, especially in the initial phase of optimization (the figure 3.1).

In the second *tune* phase, the learning rate is reduced to $\eta = 10^{-4}$ (samples once again are sorted), and is being reduced in subsequent epochs according to the equation:

$$\eta := \frac{\eta}{k^{epoch}} \tag{5.1}$$

where $k = 1.2$. The second phase aims to fine-tune parameters in the discovered stable parameters region. The presented learning rate scheduler is used unchangeable for each experiment.

## Batch Size

Most of the model optimizations converge with the *mini batch* of size over 32 samples, where the models that use the *Batch Normalization* regularization are an exception. The *Batch Normalization* is based on the batch statistics [35]. The statistics within a small batch are highly inaccurate, what in consequence, leads to the strong model regularization, and in extreme cases causes the destabilization of training. The optimization on the smallest possible convergent *mini batch* leads to better results, thanks to the vast number

of iterations. However, training is often difficult to perform due to high fluctuations and long plateaus. On the other hand, a larger batch size significantly reduces the optimization time. The computing *GPU* units are used efficiently, when the transmitted matrices for calculation are maximally large. In consequence, as the trade-off, each model is optimized with a batch size equals 256 (reduced twice in relation to [30]). It is worth noting that all samples in a single batch are aligned to the longest sample in a batch. In case of very long utterances, the batch does not fit on the *GPU* units, which unexpectedly results in the critical error (*Out Of Memory*) and stopping the calculations. The problem disappears when the samples are trimmed.

## 5.2 System Optimization

The model optimization has to be adapted to the available infrastructure on which the calculations are performed. We do the experiments on two identical computer clusters. A single experiment uses the 5 *graphical process units* (*GPU*) with the 12 GB memory.[1] Furthermore, each machine has the 46 *CPU* processors, used for the data preprocessing. In this section, we outline the three system optimization techniques, which help to do experiments effectively.

It is worth noting that the optimization time strongly depends on the number of model parameters (figure 5.2). The size of the model is one of the factors taken into consideration when exploring the different model architectures, since working with larger models limits the number of experiments.

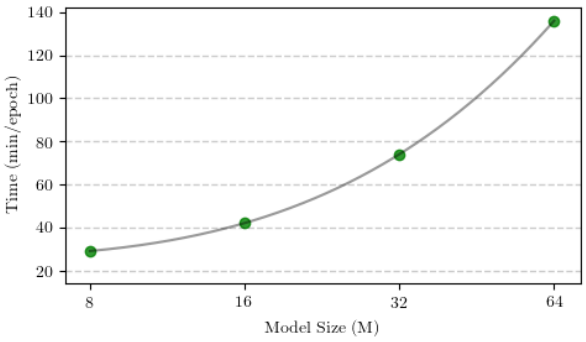


Figure 5.2: The correlation between the duration of a single epoch and the size of the model (the first series of experiments 6.1).

[1] The GPU units are the NVIDIA Tesla K80.

### Features Preprocessing

Firstly, we reduce the waiting time for the preparation input data as much as possible. The preprocessing is done parallel to the model optimization and is carried out on parallel *CPU* processes. The prepared input batches are waiting in the queue (in the RAM). Alternatively, the input data can be loaded from previously processed data, and saved in the *Hierarchical Data Frame* (*HDF*) container [14]. The container is designed to read the stored data quickly, as well in parallel processes.

### Data Parallelism

The next step is to use efficiently the available *GPU* units thanks to the *data parallelism*. The *data parallelism* replicates the model to all GPUs, where each model processes its input data portion independently. Then the partial results from various GPUs are combined on CPU into one complete batch. Our implementation allows for the quasi-linear acceleration on 5 GPUs. However, the condition is to operate on relatively large batch sizes, so that the memory of each unit is overloaded to the capacity limit.

### CuDNN-LSTM

The *LSTM* layer optimization, which is the core component of the model, is computationally expensive. In our models, the *LSTM* layer is implemented in the *CuDNN* library using the *CUDA API*. In consequence, the model training is 7 times faster than a standard implementation in the *tensorflow* library [37]. As a result, the computing cost against the basic *RNN* layer has decreased, and the benefits of using a *LSTM* layer have become more apparent [30].

# Chapter 6

# Experiments

Based on the Data, the Model and the Optimization we conducted various experiments, and the evaluation of the selected models on the *Clarin-dev* dataset are presented in this chapter. The optimization of all models is uniform and presented in detail in the Chapter Model Optimization. We use the data augmentation with the following parameters $T =< 5, 15 >$, $T_{ratio} = 0.3$, $T_{space} = 10$, $F = 20$, $m_f = 1$, which brings the most benefits, and they are the most accurate to the characteristics of the Polish language (3.3) [18]. The decoding process is also uniform for all experiments, where we use the beam of size 1024, and we do not use any external language model.

## 6.1 Base Model

We have to perform many experiments to adapt the basic model architecture to the task conditions. There are no studies on how deep neural networks *end-to-end* manage to overcome either the inflected language complexity or the highly limited training dataset. The well-explored basic model is a reliable reference point for verification, whether the presented *Synthetic Boosted Model* is an attractive approach. These series of experiments aim to verify the key components of the ASR model, which are the *Temporal Convolution*, the *Stacked Recurrent Layers* or the *Time-Frequency Convolution*. The training corpus is for each experiment the same, and contains the *Jurisdic* as well as the *Clarin-train* dataset (Chapter Data).

### Temporal Convolution

The architecture of the first model is based on the *Deep Speech* model, which is composed of five layers [25]. The first layer is a one-dimensional, in the time domain, the *Temporal*

*Convolution* layer with the filter size 15x80, and the number of filters equals to the width of the next layer. Then there are two *fully-connected* layers. Both the convolution layer and the *fully-connected* layers have the activation function *clipped rectified-linear* with a boundary value of 20.[1] The fourth layer is a bidirectional recurrent layer`LSTM`. At the end of the model, there is a linear layer with the *softmax* activation function. At the time of training, the *droupout* method [29] is applied between the *fully connected* layers to reduce the effect of *overfitting* (rate 10%).

| WER | CER | CTC Loss | Units | Model Size (M) |
|---|---|---|---|---|
| 34.87 | 7.09 | 10.23 | 650 | 8 |
| 33.13 | 6.74 | 9.55 | 900 | 16 |
| 33.85 | 6.90 | 9.85 | 1300 | 32 |
| 33.85 | 6.89 | 10.01 | 1850 | 64 |

Table 6.1: The results of the *Temporal Convolution* models in terms of the number of parameters.

The table 6.1 presents the results of the four models that have parameters between 8 and 64 million. To compare the models, we use the *Word Error Rate* metric (WER), but also we report the *Char Error Rate* (CER) metric, and the objective function values *CTC Loss*. The size of the model is exclusively modified by changing the width of all layers equally.

### Stacked Recurrent Layers

In this section, we present how *stacked recurrent* layers impact to the model performance. Similar to the *Deep Speech* model, the first layer is the *Temporal Convolution* layer with a filter dimension 15x80, and the number of filters equals to the width of the next layer. Then there are bidirectional LSTM recurrent layers. At the end of the model, there is a linear layer with an activation function *softmax*.

The table 6.2 shows the experiment results. Furthermore, our attempts to do the *Batch Normalization* in the LSTM layer before each activation function do not bring positive results [30]. Unfortunately, the effective implementation *CuDNN-LSTM*, presented in the Chapter 5.2, precludes to implement the *sequence-wise* normalization [35], which in consequence, is not explored.

[1] The activation function is limited in order to avoid the gradient exploding.

| WER | CER | CTC Loss | LSTM size | Architecture | Units (M) |
|---|---|---|---|---|---|
| 33.39 | 6.50 | 8.76 | 900 | 5-layers, 1 RNN | 16 |
| 29,05 | 5,82 | 7,79 | 550 | 5-layers | 16 |
| 24,06 | 4,78 | 6,49 | 800 | 5-layers | 32 |
| 20,73 | 4,27 | 5,73 | 650 | 7-layers | 32 |

Table 6.2: The architectures composed of the stack of recurrent layers.

### Time-Frequency Convolution

In this section, we present how the *Time-Frequency Convolution* layer impacts model performance. At the beginning of the model, there is a two-dimensional *convolution Layer*. Then there are 5 bidirectional recurrent LSTM layers with the width of 650. At the end of the model, there is a linear layer with the *softmax* activation function. In the two-dimensional convolution layers shown here, the first dimension is time and the second dimension is frequency. The width of the filter in the time domain is constant and equals 15.

| WER | CER | CTC Loss | Type | Channels | Filter dimension | Stride |
|---|---|---|---|---|---|---|
| 20,73 | 4,27 | 5,73 | 1-layer 1D | 650 | 15 | 1x1 |
| 16,60 | 3,39 | 4,80 | 1-layer 2D | 32 | 15x41 | 2x2 |
| 22,35 | 4,75 | 6,86 | 2-layer 2D | 32, 64 | 15x41, 15x21 | 2x2, 2x1 |

Table 6.3: The impact of the *Time-Frequency Convolution* layers.

The results are presented in the table 6.3 The *stride* (a step of convolution operation) is applied both in the frequency as well as in the time domain. The length of the sequence to be processed by subsequent recurrent layers is halved, which reduces the duration of the experiments almost two times.

## 6.2 Synthetic Boosted Model

In this section, we introduce two experiment series that presents the use of the *Synthetic Boosted Model* on different training datasets. In the *Entire Dataset* section, we perform experiments on the entire training dataset, which is composed of the *Jurisdic*, the *Clarin-train*, and synthetic data. Afterward, in the *Limited Dataset* section, the training corpus is strongly reduced, and only consists of the *Clarin-train* and synthetic

data.

### Entire Dataset

The *Synthetic Boosted Model* is composed of two integrated parts, the *Phoneme Model* and the *Synthetic Language Model*, therefore, training is divided into two stages (Chapter Model). Initially, we train the *Phoneme Model* (model named *Phoneme* in the table 6.4) on the training dataset composed of the *Jurisdic*, the *Clarin-train* and an additional 300 hours of synthetic data. The model contains 7 hidden layers. The first layer is the two-dimensional *Convolutional Layer*, where the receptive field size is 15x41 (time-frequency), and the number of channels equals 32. Then, there are 5 bidirectional recurrent LSTM layers with the width of 650. At the end of the model, there is a linear layer with an activation function *softmax*, which is deleted after the training. The objective function is the weighted sum of the *CTC Loss* and the *Adversarial Loss* (theirs weights are equal). The *Adversarial Loss* evaluates the hidden states of the last LSTM layer.[2]

| WER | CER | CTC Loss | Auth. Audio | Syn. Audio | Model |
|---|---|---|---|---|---|
| 16,60 | 3,39 | 4,80 | 385 | | Base |
| 15,21 | 3,12 | 4,25 | 385 | 300 | Phoneme |
| 14,33 | 2,78 | 3,99 | 385 | 1000 | Boosted |

Table 6.4: The results of the *Synthetic Language Model* where the training dataset is composed of the *Jurisdic*, the *Clarin-train* and an additional 1,000 hours of synthetic data.

The second stage is to train the *Synthetic Language Model* on the training dataset composed of the *Jurisdic*, the *Clarin-train* and an additional 1000 hours of synthetic data (model named *Boosted* in the table 6.4). The *Phoneme Model* is frozen (weights are not updated), therefore, it plays exclusively a feature extractor role at this stage. It provides input data (hidden states $h$ of the last layer LSTM) for the *Synthetic Language Model*. The *Synthetic Language Model* contains 5 hidden layers. The first layer is the *projection* layer, a narrow *fully-connected* layer, with the width of 36, and the activation function *clipped rectified-linear* with the boundary value of 20. Then there are 3 bidirectional recurrent LSTM layers with the width of 650. At the end of the model, there is a linear layer with the activation function *softmax*.

[2] We have done experiments, where the *Adversarial Loss* evaluates either the first or the third LSTM representations. However, the training is stopped on the high *plateau* due to the strong regularization.

### Limited Dataset

In these experiments, the authentic training dataset is strongly reduced, where the training corpus solely consists of the *Clarin-train* and synthetic data. Both the *Phoneme Model* and the *Synthetic Language Model*, as well as the training policy, are the same as those presented in the subsection *Entire Dataset*. We do series of experiments in which we try to customize the *Phoneme Model* to the limited dataset. In fact, the adjustments do not bring noticeable improvements.

| WER | CER | CTC Loss | Auth. Audio | Syn. Audio | Model |
|---|---|---|---|---|---|
| 21,69 | 4,22 | 5,71 | 34 | | Base |
| 21,37 | 4,20 | 5,92 | 34 | | Base+ |
| 21,13 | 4,20 | 5,66 | 34 | 34 | Phoneme |
| 19,81 | 3,96 | 5,67 | 34 | 1000 | Boosted |
| 16,60 | 3,39 | 4,80 | 385 | | Base |

Table 6.5: The results of the *Synthetic Language Model* where the training dataset is composed of the *Clarin-train* and an additional 1000 hours of synthetic data.

The results are presented in the table 6.5. To the comparison, we present the *Base+* model, which has the same architecture as the *Boosted* model (and use the frozen *Base* model), but it is trained exclusively on the *Clarin-train* dataset.

## 6.3 Conclusions

### Base Model

We began our experiments with the model architecture based on the *Deep Speech* model. The model with 16 million parameters achieves the best result, where the WER equals 33.13%. The number of model parameters strongly depends on the size of the training dataset. Our dataset is much smaller than the one presented in [25], therefore, we achieve similar results, even when we reduce the number of parameters eight times beside the *Deep Speech* model. The table 6.6 shows the selected examples of model predictions. Despite the high WER level, it is apparent that the model has learned different spelling rules, for instance *zdążyłem*, *żadnej*, and *alkoholiku*. However, the WER score above 30% is too high to make the system useful.

In the *Temporal Convolution* models, the first three layers build a representation solely on a small context of a signal. The broader context, done by replacing fully connected

| Model prediction | Correct transcription |
|---|---|
| ja nie zdążyłem rozpocząć żadnej działalności a**l**i | ja nie zdążyłem rozpocząć żadnej działalności a**n**i |
| słowami takimi jak brydziarzu złodzieju alkoholiku | słowami takimi jak bry**n**dziarzu złodzieju alkoholiku |
| dowodowymi i **określe nia** wartości poszczególnych dowodu | dowodowymi i **określenia** wartości poszczególnych dowodów |

Table 6.6: The selected predictions of the *Temporal Convolution* model with the layer width 900 (16 million parameters).

layers to recurrent layers, brings better performance, what we presented in the *Stacked Recurrent Layers* experiments. The best model architecture has 32 million parameters and consists of 7 layers, including 5 `LSTM` layers with the width of 650. Although the model is composed of 7 layers (a gradient propagation travels a long way), the model optimization is stable thanks to the data augmentation technique, the large batch size (256), and uniform sample lengths from 3 to 7 words (Chapter Data). We are aware that by doing so, we lose the ability to represent contexts longer than 7 words.

In *Time-Frequency Convolution* experiments, we present that the use of a single two-dimensional convolutional layer significantly improves model generalization (the WER score drops to 16.60%). The one-dimensional (time) convolutional layer has a large perception area (15x80) and is non-resistant to shifts in the frequency domain. We analyze the filters of the one-dimensional convolutional layer, which shows that most of the filters are unused. The used filters are grouped into several types (similar shapes), and they are concentrated in tiny areas. A two-dimensional convolutional layer accurately resolves these issues. It groups filters of the similar shape (the number of *channels* is reduced), which are shifted in the frequency domain.

## Synthetic Boosted Models

The experiments presented in the section 6.2 confirms that the large synthetic dataset can be successfully used in *end-to-end* speech recognition models. The *Synthetic Boosted Model* trained on the entire synthetically augmented dataset achieves more than 13% absolute better the WER score, which equals 14,33%.

In the figure 6.1, we present two matrices. On the left is the matrix grouping mistakes of the *Synthetic Boosted Model* by the *Char Edit Distance* and the *Word Edit Distance*. We can see that more than 800 samples have a single spelling mistake. The most dif-

ficult to correct mistakes are on the out of a diagonal, which have high values of the *Char Edit Distance*. On the right side, we see the corrected mistakes with respect to the *Phoneme Model*. We can observe that the *Synthetic Language Model* also corrects mistakes that have the high *Char Edit Distance* value.

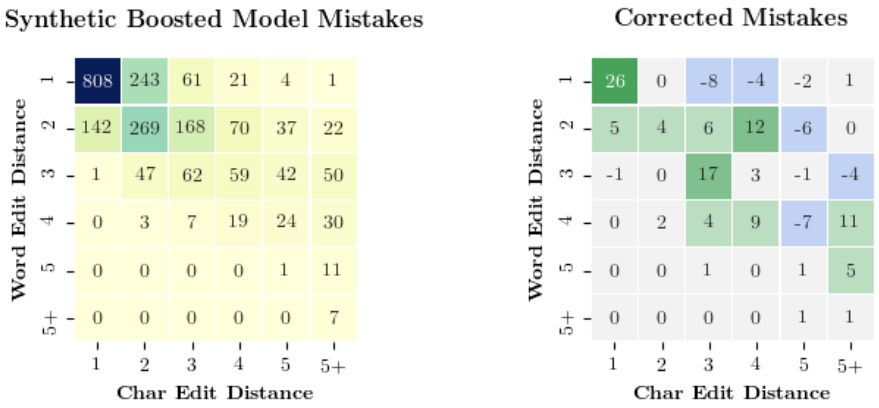


Figure 6.1: On the left, the grouped mistakes of the *Synthetic Boosted Model* by the *Char Edit Distance* and the *Word Edit Distance*. On the right, the corrected mistakes, which are defined as the difference between the *Phoneme Model* and the *Synthetic Boosted Model*.

The *Synthetic Boosted Model* trained on the *Limited Dataset* also outperforms the *Base Model* with almost 10% absolute better WER score, which equals 19.81%. The difference in the performance of two *base models*, which are trained on 34 and 385 hours of authentic data, equals 5.09 of WER score. The *Synthetic Boosted Model* covers 36.94% of this difference using 1000 hours of synthetic data. We are aware that there could be many factors which cause the inefficient use of synthetic data. Nonetheless, we introduce further one of them.

We do a test to check the linear separability of the representation $h$ returned by the *Phoneme Model*. From the training corpus, we selected a thousand samples of authentic and synthetic data (the ratio 1 to 1). Then, we process these samples using the *Phoneme Model*, in a purpose to collect the hidden states $h$ for each LSTM layer. Finally, we build a set of simple binary classifiers, which try to predict whether the representation $h$ comes from authentic or synthetic data (different classifier for each layer).[3]

The classifier that analyzes the activation of the first LSTM layer, immediately and error-free recognizes the activation source (figure 6.2). In accordance with our expectations, subsequent classifiers have larger difficulties in identifying the data source. On

[3] The classifiers are the same, and they are composed of one neuron with the activation function *sigmoid*.

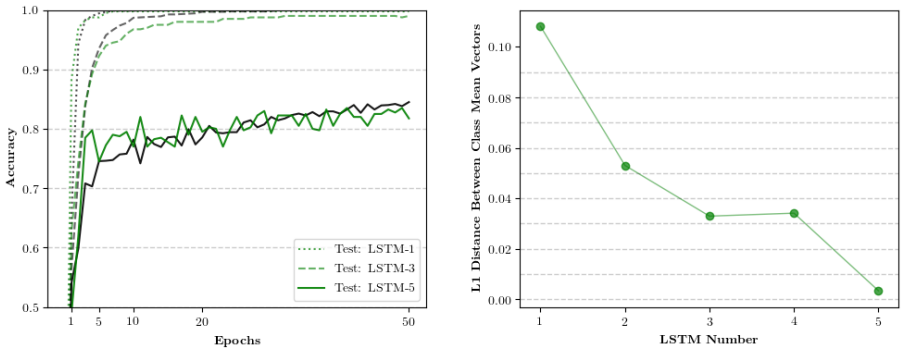


Figure 6.2: The linear separability test of the representations $h$ returned by the *Phoneme Model*. Classifiers try to predict whether the representations $h$ comes from authentic or synthetic data.

the right-hand side, the graph presents the values of the *Adversarial Loss* depending on the depth of the LSTM layer. We can see that despite the near-zero loss function value (the fifth layer), the linear classifier separates with the high precision the activations from authentic and synthetic data (the *accuracy* over 80%). The further investigation of the other *Adversarial Loss* function is highly recommended.

# Chapter 7

# Evaluation

We do the evaluation on the hold-out *Clarin-test* dataset, which has independent speakers and unique transcriptions (Chapter Data). The architecture of both models is presented in the section Synthetic Boosted Model. The *Synthetic Boosted Model* (the model named *Boosted* in the table 7.1) trained on the entire synthetically augmented dataset achieves more than 12.5% absolute better the WER score, which equals 14,53%. The results on the *Clarin-dev* dataset are comparable.

| WER-test | CER-test | WER-dev | CER-dev | Auth. Audio | Syn. Audio | Model |
|---|---|---|---|---|---|---|
| 16,64 | 3,34 | 16,60 | 3,39 | 385 | | Base |
| 14,53 | 2,98 | 14,33 | 2,78 | 385 | 1000 | Boosted |

Table 7.1: The final evaluation results on the *Clarin-test* dataset, and as the reference point the results on the *Clarin-dev* dataset (Chapter Model).

In the figure 6.1, we present two matrices. On the left is the matrix grouping mistakes of the *Synthetic Boosted Model* by the *Char Edit Distance* and the *Word Edit Distance* (as these presented in the section 6.3). We can see that more than 800 samples have a single spelling mistake. On the right side, we see the corrected mistakes with respect to the *Base Model*. The most difficult to correct mistakes are on the out of a diagonal, which have high values of the *Char Edit Distance*. We can observe that the *Synthetic Language Model* also corrects mistakes that have high *Char Edit Distance* values.

### Error Analysis

We do the basic error analysis of the *Synthetic Boosted Model*. At the beginning, we present how the char edit distance (*Levenshtein distance* on the char level) is distributed

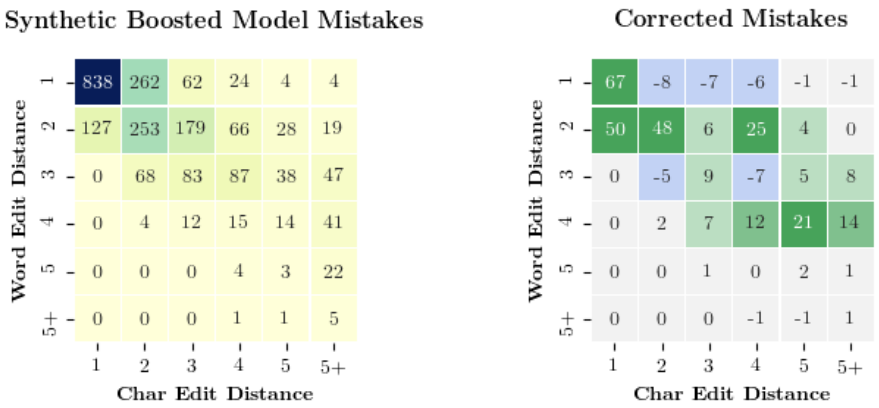


Figure 7.1: On the left, the grouped mistakes of the *Synthetic Boosted Model* by the *Char Edit Distance* and the *Word Edit Distance*. On the right, the corrected mistakes, which are defined as the difference between the *Base Model* and *Synthetic Boosted Model*.

among different operations: the *insertion*, the *deletion*, and the *substitution*. The distribution is presented in the figure 7.2. The deletion operations are more than two times less than either the insertion or the substitution. The model tends to predict a char, only while it is confident.

Moreover, inside the donut chart, we mark the share of mistakes caused by the *space* char, where it should be added or deleted. Over 15% of all mistakes are caused by the *space*, which has a high impact on the WER score. In fact, only a single misspelled space in the sequence composed of four words brings up to 50% of the WER score.

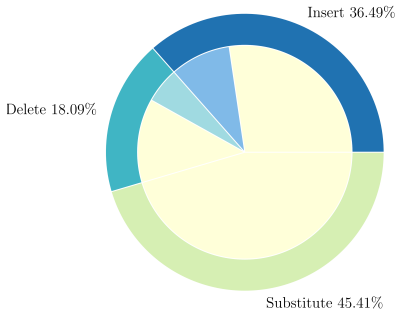


Figure 7.2: The char edit distance distributed among different operations: the insertion, the deletion, and the substitution.

The largest group of misspellings is caused by the wrong character prediction, which represents the substitution operation. To show the mistakes in detail, we present the confusion matrix in the figure 7.3. The overall pattern is that vowels and consonants are

grouped. There are various concrete patterns which can be easily interpreted. One of them is that the chars frequently occurred at the end of words 'a', 'e', 'y' are often each other misspelled. The letters which sound similar, but depend on the context are written differently, such as t and d, s and z, or p and b. Finally, the different grammar rules, for instance, the u and ó, the j and i are hard to be correctly defined.

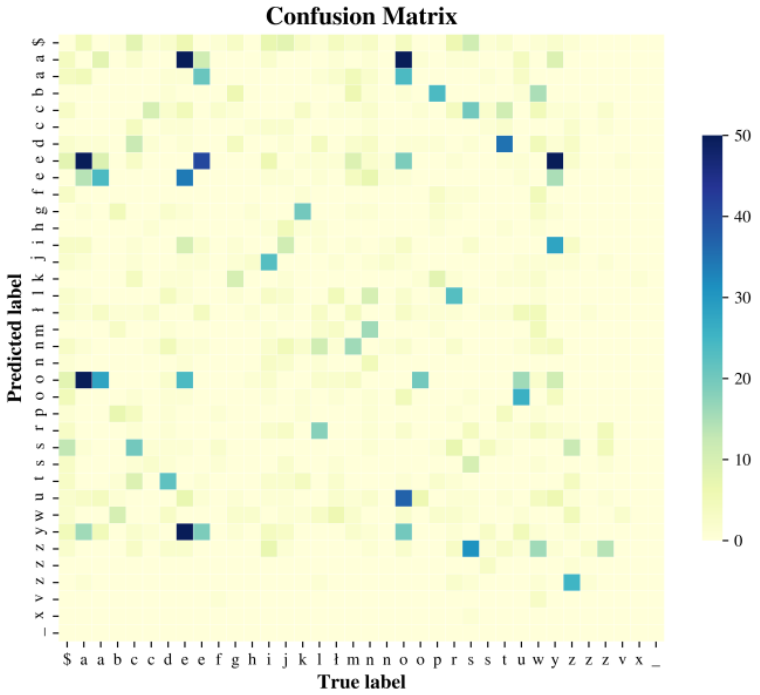


Figure 7.3: The confusion matrix of misspelled characters.

# Chapter 8

# Conclusion and Future Works

The presented *Synthetic Boosted Model* has successfully improved the performance of the automatic speech recognition *end-to-end* neural network model. The vast amount of synthetic data can be used to increase the language information supplied to the model. The improvement is achieved thanks to the new model architecture, the new objective function, and the new training policy. However, each component in the presented *Synthetic Boosted* approach can be further improved. To start exploring further ideas, we encourage to visit and benefit from our open-source project Synthetic Boosted DeepSpeech, where one can find the implementation of the entire system presented in this work, and more. The pre-trained models are publicly available and ready to use.

**Model Architecture**

The *Synthetic Boosted* approach can be further explored independently to the model architecture. The research of using the *Synthetic Boosted* method in the *Sequence-to-Sequence* models should be profoundly investigated.

**Adversarial Component**

In our work, we introduce the naive adversarial component as the part of the new objective function. However, we deeply believe that more accurate function exist, and should be applied.

**Specialized Domain Adaptation**

The *Synthetic Boosted* approach can be used as an auxiliary data augmentation method. This method could turn out to be accurate to perform the specialized domain adaptation. The *Synthetic Boosted* approach should be tested on different languages and specialized datasets.

# List of Tables

# List of Figures

# Bibliography


[1] S. S. Stevens, J. Volkmann, and E. B. Newman, "A scale for the measurement of the psychological magnitude pitch", *Journal of the Acoustical Society of America*, vol. 8, pp. 185–190, 1937, ISSN: 0001-4966(Print). DOI: 10.1121/1.1915893.

[2] B. T. Lowerre, "The Harpy Speech Recognition System.", PhD Thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1976.

[3] J. Harris, "On then Use of Windows for Harmonic Analysis with the Discrete Fourier Transform", p. 33, 1978.

[4] H. A. Bourlard and N. Morgan, *Connectionist Speech Recognition: A Hybrid Approach*. Norwell, MA, USA: Kluwer Academic Publishers, 1993, ISBN: 978-0-7923-9396-2.

[5] S. Renals, N. Morgan, H. Bourlard, *et al.*, "Connectionist probability estimators in HMM speech recognition", *IEEE Trans. Speech and Audio Processing*, vol. 2, pp. 161–174, 1994. DOI: 10.1109/89.260359.

[6] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory", *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, ISSN: 0899-7667. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735 (visited on 10/11/2019).

[7] D. Ellis and N. Morgan, "Size matters: An empirical study of neural network training for large vocabulary continuous speech recognition", in *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258)*, vol. 2, Mar. 1999, 1013–1016 vol.2. DOI: 10.1109/ICASSP.1999.759875.

[8] J. R. J. Deller, J. H. L. Hansen, and J. G. Proakis, *Discrete-Time Processing of Speech Signals*. Wiley, 2000, 944 pp., ISBN: 978-0-7803-5386-2.

[9] B. H. Juang and L. R. Rabiner, "Automatic Speech Recognition – A Brief History of the Technology Development Abstract", 2004.

[10] A. Graves, S. Fernández, F. Gomez, *et al.*, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural 'networks", presented at the ICML 2006 - Proceedings of the 23rd International Conference on Machine Learning, vol. 2006, Jan. 1, 2006, pp. 369–376. DOI: 10.1145/1143844.1143891.

[11] G. Demenko, S. Grocholewski, K. Klessa, *et al.*, "JURISDIC: Polish Speech Database for Taking Dictation of Legal Texts", in *LREC*, 2008.

[12] T. Váradi, S. Krauwer, P. Wittenburg, *et al.*, "CLARIN: Common Language Resources and Technology Infrastructure", in *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco: European Languages Resources Association (ELRA), May 2008.

[13] G. E. Dahl, D. Yu, L. Deng, *et al.*, "Large vocabulary continuous speech recognition with context-dependent DBN-HMMS", in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2011, pp. 4688–4691. DOI: 10.1109/ICASSP.2011.5947401.

[14] M. Folk, G. Heber, Q. Koziol, *et al.*, "An Overview of the HDF5 Technology Suite and Its Applications", in *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, (Uppsala, Sweden), ser. AD '11, New York, NY, USA: ACM, 2011, pp. 36–47, ISBN: 978-1-4503-0614-0. [Online]. Available: http://doi.acm.org/10.1145/1966895.1966900.

[15] O. Abdel-Hamid, A. Mohamed, H. Jiang, *et al.*, "Applying Convolutional Neural Networks concepts to hybrid NN-HMM model for speech recognition", in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2012, pp. 4277–4280. DOI: 10.1109/ICASSP.2012.6288864.

[16] M. Sahidullah and G. Saha, "Design, analysis and experimental evaluation of block based transformation in MFCC computation for speaker recognition", *Speech Communication*, vol. 54, no. 4, pp. 543–565, May 1, 2012, ISSN: 0167-6393. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167639311001622.

[17] A. Ghoshal and D. Povey, "Sequence discriminative training of deep neural networks", in *In Proc. INTERSPEECH*, 2013.

[18] M. Igras and B. Zio łko, "Length of Phonemes in a Context of their Positions in Polish Sentences", in *Proceedings of the 10th International Conference on Signal Processing and Multimedia Applications and 10th International Conference on Wireless Information Networks and Systems*, Reykjavík, Iceland: SCITEPRESS - Science and and Technology Publications, 2013, pp. 59–64, ISBN: 978-989-8565-74-7.

[Online]. Available: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0004503500590064.

[19] N. Kanda, R. Takeda, and Y. Obuchi, "Elastic spectral distortion for low resource speech recognition with deep neural networks", in *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, Dec. 2013, pp. 309–314. DOI: 10.1109/ASRU.2013.6707748.

[20] T. Mikolov, I. Sutskever, K. Chen, *et al.*, "Distributed Representations of Words and Phrases and their Compositionality", *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, *et al.*, Eds., pp. 3111–3119, 2013. [Online]. Available: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf (visited on 10/11/2019).

[21] T. N. Sainath, A. Mohamed, B. Kingsbury, *et al.*, "Deep convolutional neural networks for LVCSR", in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2013, pp. 8614–8618. DOI: 10.1109/ICASSP.2013.6639347.

[22] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate", Sep. 1, 2014. arXiv: 1409.0473 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1409.0473.

[23] K. Cho, B. van Merriënboer, C. Gulcehre, *et al.*, "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation", in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. DOI: 10.3115/v1/D14-1179.

[24] A. Graves and N. Jaitly, "Towards End-to-end Speech Recognition with Recurrent Neural Networks", in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, (Beijing, China), ser. ICML'14, JMLR.org, 2014, pp. II-1764–II-1772. [Online]. Available: http://dl.acm.org/citation.cfm?id=3044805.3045089.

[25] A. Hannun, C. Case, J. Casper, *et al.*, "Deep Speech: Scaling up end-to-end speech recognition", pp. 1–12, 2014. [Online]. Available: http://arxiv.org/abs/1412.5567.

[26] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", Dec. 22, 2014. arXiv: 1412.6980 [cs]. [Online]. Available: http://arxiv.org/abs/1412.6980 (visited on 10/10/2019).

[27] A. Ragni, K. Knill, S. P. Rath, *et al.*, "Data augmentation for low resource languages", in *INTERSPEECH*, 2014.

[28] H. Soltau, G. Saon, and T. N. Sainath, "Joint training of convolutional and non-convolutional neural networks", in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2014, pp. 5572–5576. DOI: 10.1109/ICASSP.2014.6854669.

[29] N. Srivastava, G. Hinton, A. Krizhevsky, *et al.*, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html.

[30] D. Amodei, R. Anubhai, E. Battenberg, *et al.*, "Deep Speech 2: End-to-End Speech Recognition in English and Mandarin", pp. 1–28, 2015. [Online]. Available: http://arxiv.org/abs/1512.02595.

[31] W. Chan, N. Jaitly, Q. V. Le, *et al.*, "Listen, Attend and Spell", Aug. 5, 2015. arXiv: 1508.01211 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1508.01211.

[32] Ç. Gülçehre, O. Firat, K. Xu, *et al.*, "On Using Monolingual Corpora in Neural Machine Translation", *ArXiv*, vol. abs/1503.03535, 2015. arXiv: 1503.03535.

[33] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", in *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, (Lille, France), ser. ICML'15, JMLR.org, 2015, pp. 448–456. [Online]. Available: http://dl.acm.org/citation.cfm?id=3045118.3045167 (visited on 10/11/2019).

[34] T. Ko, V. Peddinti, D. Povey, *et al.*, "Audio augmentation for speech recognition", in *INTERSPEECH*, 2015.

[35] C. Laurent, G. Pereyra, P. Brakel, *et al.*, "Batch Normalized Recurrent Neural Networks", Oct. 5, 2015. arXiv: 1510.01378 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1510.01378.

[36] C. Olah. (2015). Understanding LSTM Networks, [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[37] M. Abadi, P. Barham, J. Chen, *et al.*, "TensorFlow: A System for Large-scale Machine Learning", in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, (Savannah, GA, USA), ser. OSDI'16, Berkeley, CA, USA: USENIX Association, 2016, pp. 265–283, ISBN: 978-1-931971-33-1. [On-

line]. Available: http://dl.acm.org/citation.cfm?id=3026877.3026899 (visited on 10/11/2019).

[38] J. Chorowski and N. Jaitly, "Towards better decoding and language model integration in sequence to sequence models", in *INTERSPEECH*, 2016. DOI: 10.21437/interspeech.2017-343. arXiv: 1612.02695.

[39] A. van den Oord, S. Dieleman, H. Zen, *et al.*, "WaveNet: A Generative Model for Raw Audio", Sep. 12, 2016. arXiv: 1609.03499 [cs]. [Online]. Available: http://arxiv.org/abs/1609.03499.

[40] J. P. Ryan. (2016). Around the World in 60 Days: Getting Deep Speech to Work in Mandarin, [Online]. Available: https://svail.github.io/mandarin/.

[41] R. Chaabouni, E. Dunbar, N. Zeghidour, *et al.*, "Learning Weakly Supervised Multimodal Phoneme Embeddings", in *Interspeech 2017*, ISCA, Aug. 20, 2017, pp. 2218–2222. [Online]. Available: http://www.isca-speech.org/archive/Interspeech_2017/abstracts/1689.html.

[42] A. Hannun, "Sequence Modeling with CTC", *Distill*, vol. 2, no. 11, e8, Nov. 27, 2017, ISSN: 2476-0757. [Online]. Available: https://distill.pub/2017/ctc.

[43] D. Koržinek, K. Marasek, Ł. Brocki, *et al.*, "Polish Read Speech Corpus for Speech Tools and Services", Jun. 1, 2017. arXiv: 1706.00245 [cs]. [Online]. Available: http://arxiv.org/abs/1706.00245.

[44] Y.-A. Chung and J. Glass, "Speech2Vec: A Sequence-to-Sequence Framework for Learning Word Embeddings from Speech", in *Interspeech 2018*, ISCA, Sep. 2, 2018, pp. 811–815. [Online]. Available: http://www.isca-speech.org/archive/Interspeech_2018/abstracts/2341.html.

[45] J. Howard and S. Ruder, "Universal Language Model Fine-tuning for Text Classification", in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 328–339. DOI: 10.18653/v1/P18-1031.

[46] O. Kuchaiev, B. Ginsburg, I. Gitman, *et al.*, "OpenSeq2Seq: Extensible toolkit for distributed and mixed precision training of sequence-to-sequence models", *ArXiv*, vol. abs/1805.10387, 2018. DOI: 10.18653/v1/w18-2507.

[47] J. Li, R. Gadde, B. Ginsburg, *et al.*, "Training Neural Speech Recognition Systems with Synthetic Speech Augmentation", Nov. 2018. [Online]. Available: http://arxiv.org/abs/1811.00707.

[48] M. Ogrodniczuk and Ł. Kobyliński, Eds., *Proceedings of the PolEval 2018 Workshop (Task 3: Language Models)*, Warsaw, Poland: Institute of Computer Science, Polish Academy of Sciences, 2018, ISBN: 978-83-63159-27-6. [Online]. Available: http://2018.poleval.pl/index.php/tasks/.

[49] M. Peters, M. Neumann, M. Iyyer, *et al.*, "Deep Contextualized Word Representations", in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, New Orleans, Louisiana: Association for Computational Linguistics, Jun. 2018, pp. 2227–2237. DOI: 10.18653/v1/N18-1202.

[50] A. Sriram, H. Jun, S. Satheesh, *et al.*, "Cold fusion: Training Seq2seq models together with language models", *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, vol. 2018-Septe, pp. 387–391, 2018, ISSN: 19909772. DOI: 10.21437/Interspeech.2018-1392.

[51] J. Devlin, M.-W. Chang, K. Lee, *et al.*, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", in *NAACL-HLT*, 2019. arXiv: 1810.04805.

[52] M. El-Geish, "Learning Joint Acoustic-Phonetic Word Embeddings", *ArXiv*, vol. abs/1908.00493, 2019. arXiv: 1908.00493.

[53] Y. Jia, M. Johnson, W. Macherey, *et al.*, "Leveraging Weakly Supervised Data to Improve End-to-end Speech-to-text Translation", in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2019, pp. 7180–7184. DOI: 10.1109/ICASSP.2019.8683343.

[54] J. Li, V. Lavrukhin, B. Ginsburg, *et al.*, "Jasper: An End-to-End Convolutional Neural Acoustic Model", 2019. arXiv: 1904.03288 [cs, eess]. [Online]. Available: http://arxiv.org/abs/1904.03288.

[55] D. S. Park, W. Chan, Y. Zhang, *et al.*, "SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition", Apr. 2019. [Online]. Available: http://arxiv.org/abs/1904.08779.

[56] N. Zeghidour, "Learning representations of speech from the raw waveform", Mar. 13, 2019. [Online]. Available: https://tel.archives-ouvertes.fr/tel-02278616.