Georgescu *et al. EURASIP Journal on Audio, Speech, and Music Processing* (2021) 2021:28
https://doi.org/10.1186/s13636-021-00217-4

EURASIP Journal on Audio, Speech, and Music Processing

REVIEW Open Access

Performance vs. hardware requirements in state-of-the-art automatic speech recognition

Alexandru-Lucian Georgescu[1*], Alessandro Pappalardo[2], Horia Cucu[1] and Michaela Blott[2]

Abstract

The last decade brought significant advances in automatic speech recognition (ASR) thanks to the evolution of deep learning methods. ASR systems evolved from pipeline-based systems, that modeled hand-crafted speech features with probabilistic frameworks and generated phone posteriors, to end-to-end (E2E) systems, that translate the raw waveform directly into words using one deep neural network (DNN). The transcription accuracy greatly increased, leading to ASR technology being integrated into many commercial applications. However, few of the existing ASR technologies are suitable for integration in embedded applications, due to their hard constrains related to computing power and memory usage. This overview paper serves as a guided tour through the recent literature on speech recognition and compares the most popular ASR implementations. The comparison emphasizes the trade-off between ASR performance and hardware requirements, to further serve decision makers in choosing the system which fits best their embedded application. To the best of our knowledge, this is the first study to provide this kind of trade-off analysis for state-of-the-art ASR systems.

Keywords: Automatic speech recognition, Survey, End-to-end ASR systems, Deep learning, Performance analysis

1 Introduction

Speech is one of the most important forms of human communication and, given this fact, it has always been desired to extend the voice interaction with the technologies that surround us, making it as naturally as possible. In this way, the technology could be used on a larger scale, without the need for additional knowledge and even by people with disabilities or people whose activity requires hands-free operation of the devices. Automatic speech recognition (ASR) enables users to transcribe streams of speech into the corresponding texts. The last few years have brought a significant increase in this field, the voice-activated devices being particularly successful. For instance, smart-home assistants, such as Amazon Echo or virtual personal assistants like Siri, have become well-known and they are capable of performing complex speech to text tasks. They successfully transcribe even if the environmental conditions are challenging, like background noise or hesitations. Their input is represented by free speech with a high degree of variability from one speaker to another. The type of speech is usually free speech, the only constraint being the pronunciation of a keyword, which will wake up the device. Because ASR systems are highly computational and they require a large number of resources, it is very difficult for this entire process to take place in an embedded environment. Usually, hot-word detection is performed on the device, while the rest of the speech is transferred into the cloud for transcription.

However, there is an increased interest in developing speech recognition systems that can be deployed in embedded systems [1–5]. These are usually integrated in wearable devices or Internet of Things applications, thus eliminating the need for cloud processing. Embedded devices come with hard constrains related to computing

*Correspondence: lucian.georgescu@speed.pub.ro
[1]Speech and Dialogue Research Laboratory, University Politehnica of Bucharest, Bucharest, Romania
Full list of author information is available at the end of the article

Springer Open

© The Author(s). 2021 **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

power and memory usage. In this situation it is crucial to use ASR systems that allow for real-time processing, along with very low energy consumption and low memory footprint, but at the same time preserving as much as possible the high accuracy of the neural network-based systems. In order to meet the requirements of embedded systems, one can resort to several optimization techniques: (i) architecture optimization, (ii) data quantization and format optimization.

Architecture optimization involves reducing the number of layers and the sizes of the layers, etc. aiming to eventually obtain a smaller neural network in terms of total number of parameters. This, along with reducing the number of activations that need to be kept in memory, leads to a system that can better fit into memory-constrained systems. Moreover, by reducing the model size one implicitly reduces the number of operations performed during inference and thus obtains a better real-time factor. Data quantization refers to the process of reducing the bit precision of the weights and activations, while data format optimization concerns using fixed-point or integer numbers instead of typical floating-point values [6]. Data quantization and format optimization help at obtaining a smaller footprint for the models and activations (e.g. 8-bit values instead of 32-bit values) and also for reducing the real-time factor by *simplifying* the operations performed during inference (e.g. multiply and accumulations with 8-bit integers instead of 32-bit floating-point values).

In this context, it is important to pinpoint the ASR systems which are suitable for deployment in embedded applications. There are several surveys which present some of the state of the art ASR systems, as follows. Zhang et al. [7] present a review focused on deep learning aiming to improve the environmental robustness of speech recognition systems. They introduce technologies related to the front-end of the systems, which is performing the signal preprocessing, but also about the back-end, which is performing the model training, for this particular subtask. The authors compare the traditional, probabilistic, systems with those based on deep learning. They provide a comparison in terms of the accuracy of some different general approaches on 4 benchmark speech databases. Park et al. [8] present real-time speech recognition systems for mobile and embedded devices. Several neural network based acoustic models are compared by accuracy versus model size. The authors create an acoustic model by combining recurrent units and convolutional layers. The main purpose is to solve the excessive memory consumption due the parameters size in recurrent neural networks by using a parallelization approach. Purwins et al. [9] present an overview of deep learning techniques used for audio signal processing. The authors make a classification of the main applications in the field of audio signal processing, highlighting the types of acoustic features, the types of neural networks used to train the model, the data requirements and the computational complexity. Kim et al. [10] describe in detail various components and algorithms used in end-to-end speech recognition systems and compare them from the architectural point of view and also in terms of accuracy. Several methods of model compression are discussed, considering the possibility of using such systems in commercial on-device applications.

As opposed to the survey papers mentioned above, which present overviews on text-to-speech systems and focus on high-level architectures, this article analyzes many concrete ASR implementations available in the most popular ASR toolkits. The various component parts of the ASR implementations are presented in detail, along with graphical representations of the underlying neural networks. Finally, we offer insightful comparisons on hardware performance versus transcription accuracy.

This survey provides an overview of the best and most popular speech recognition architectures, including architectural insights that can be used to identify the system which is most suitable for deployment in embedded applications. Data quantization and format optimization, which are not covered by this survey, can be potentially applied regardless of the selected architecture, as they are orthogonal optimizations, leading to a reduction in model size and real-time factor along with a compromise related to accuracy.

The aim of the survey is two-fold: (i) to serve as a guided tour through the recent literature on automatic speech recognition and (ii) to provide an analysis of the trade-off between ASR performance and hardware requirements for the most popular ASR implementations. The survey describes the various components and sub-systems of generic ASR systems in Section 2 and then describe, evaluates and compares eight specific ASR implementations in Sections 3 and 4. The comparison is structured as a trade-off between transcription accuracy and system complexity. We first compare the models in terms of model size, number of activations and number of operations required to process one frame of speech and then translate these metrics into hardware requirements regarding memory load and throughput. The aim is to provide a complete picture of the trade-offs between complexity and performance to further serve the decision makers in choosing the system which is most suitable for deployment in embedded applications.

The paper is organized as follows. Section 2 presents the general characteristics of a speech recognition system, offering details about its components. The differences between a traditional pipeline system and an end-to-end system are discussed. We describe the most common types of speech features and we provide information regarding different speech recognition system

architectures grouped into traditional, architectures based on Hidden Markov Models (HMM) and end-to-end neural networks, along with the most common language modeling methods. Section 3 is concerned with the description from the architectural point of view of eight of the best performing open-source speech recognition systems, while the Section 4 deals with the comparison and evaluation by considering the accuracy and the complexity of the systems on a standard speech recognition task. Section 5 is reserved for conclusions. The paper organization is presented in Table 1.

2 Introduction to ASR systems

This section introduces the basic concepts in automatic speech recognition. Section 2.1 presents the road from traditional ASR to end-to-end ASR. Section 2.2 describes the most common speech features which are used in current state-of-the-art implementations. Section 2.3 introduces the main principles in traditional ASR, while Section 2.4 presents different end-to-end approaches. Section 2.5 summarizes the common language models and their integration techniques in ASR systems.

2.1 The road from pipeline ASR to end-to-end ASR

Table 2 presents the main differences between these two categories of ASR systems in terms of architecture, decoding strategy, input and output data. A pipeline ASR involves multiple models, each one created in the training stage: an acoustic model (AM) - usually implemented as a neural network, a phonetic model (PD) and a language model (LM) - usually implemented as a probabilistic component. The decoding graph is a weighted finite state transducer (WFST), obtained by composition operations applied on these elements. The acoustic modeling element outputs posterior probabilities for context dependent phones. The phonetic modeling element combines these phones to form valid words, while the linguistic modeling element combines words in sequences as

Table 1 Paper summary

Section	Content
1. Introduction	Paper context; paper goals; paper structure
2. Introduction to ASR systems	Main concepts about the automatic speech recognition field
2.1 The road from pipeline ASR to end-to-end ASR	Differences between those two categories of systems
2.2 Feature extraction	Most popular speech features
2.3 Traditional, HMM-based acoustic modeling	Acoustic modeling using Hidden Markov Models: concepts and systems
2.4 End-to-end ASR systems	Most common end-to-end approaches
2.5 Language Modeling	Most common language modeling approaches
3. State-of-the-art ASR implementations	Detailed description of 8 speech recognition systems
3.1 Kaldi chain model TDNN	Simple time-delay neural network system
3.2 Kaldi chain model CNN-TDNN	Convolutional + time-delay neural network system
3.3 Paddle Paddle implementation of DeepSpeech2	Simple recurrent neural network system
3.4 RWTH RETURNN	Attention-based encoder-decoder neural network system
3.5 Facebook CNN-ASG	Fully convolutional with gated linear units neural network system
3.6 Facebook TDS-S2S	Convolutional with time-depth separable blocks neural network system
3.7 Nvidia Jasper	Convolutional neural network with residual connections system
3.8 Nvidia QuartzNet	Lightweight convolutional neural network with time-channel separable residual blocks system
4. ASR comparison and evaluation. Case study on LibriSpeech	Accuracy and hardware requirements of those 8 implementations evaluated on LibriSpeech task
4.1 Evaluation of model complexity	Definition of the metrics used for model complexity
4.2 Comparison of ASR systems in terms of model complexity	Complexity of the models computed as the number of parameters, operations and activations
4.3 Comparison of ASR systems in terms of performance	Transcription accuracy on LibriSpeech dataset
4.4 Trade-offs between ASR performance and hardware	Accuracy vs. hardware requirements: trade-off analysis
5. Conclusion	Paper summary; achieved goals
	Main conclusions emerged from the analysis of those 8 systems

Table 2 Main differences between pipeline ASR and end-to-end ASR in terms of architecture, decoding strategy, input and output data

	Pipeline ASR	**End-to-end ASR**
System architecture	multi-component:	single component:
	acoustic model (usually neural network)	neural network for both acoustic and language modeling
	language model (usually probabilistic)	+ optional more complex language model
	phonetic dictionary	
Decoding strategy	Weighted Finite State Transducer (WFST) decoder	Connectionist Temporal Classification (CTC) decoder
		Sequence-to-sequence attention decoder
Input	always hand-crafted features	raw waveform or sometimes hand-crafted features
	+ optional learned features	
Output	context dependent phones and then words	characters or word-parts

likely to form sentences. On the other hand, end-to-end ASR trains a single network to learn the speech representations. A connectionist temporal classification (CTC) decoder is usually used, this being a beam search based mechanism. It provides a probability distribution over the output labels set for each input time step. The CTC decoding outputs characters separated by blanks, aligned with the input sequence. Because in this way words and word sequences are directly obtained, the language model is optional in this approach. However, it ensures valid outgoing words and more likely word sequences.

2.1.1 Pipeline ASR

This represents the traditional ASR system (see the top of Fig. 1), made up of multiple components that work together as in a pipeline system. The automatic speech recognition task consists in identifying the most probable words sequence W^*, given the probability of the speech signal X to be generated by the sequence of words W:

$$W^* = \arg\max\ p(W|X). \tag{1}$$

Following the Bayes' rule, this equation could be transformed into an equivalent form:

$$\begin{aligned} W^* &= \arg\max\ p(W|X) = \arg\max \frac{P(X|W)P(W)}{p(X)} \\ &= \arg\max\ P(X|W)P(W). \end{aligned} \tag{2}$$

$P(X)$ does not depend on the words sequence W, therefore the denominator could be eliminated and the problem could be limited to the following aspects; (i) the computation of the probability of the speech signal X, given the corresponding words sequence W: $P(X|W)$ and (ii) the computation of the probability for the words sequence: $P(W)$.

The first probability can be determined using an acoustic model, while the second probability is obtained using a language model. The link between these models is provided by the phonetic model, which consists in a dictionary associating each word from the language model to a sequence of phonemes, modeled by the acoustic model.

The acoustic model estimates the likelihood that a speech signal has been generated by a sequence of words. To achieve that, it is necessary to make observations based on an extended set of word pronunciations from as many speakers as possible, then obtaining a mathematical model that estimates the probability of each word. This premise leads to the choice of the word as the base speech modeling unit, which, however, opposes a few principles. The base unit must be precise and its representation should include the manifestation in several acoustic contexts. At the same time, it must be trainable, there must exist enough data to be able to estimate the unit correctly. Last but not least, the base unit must be generalizable, being

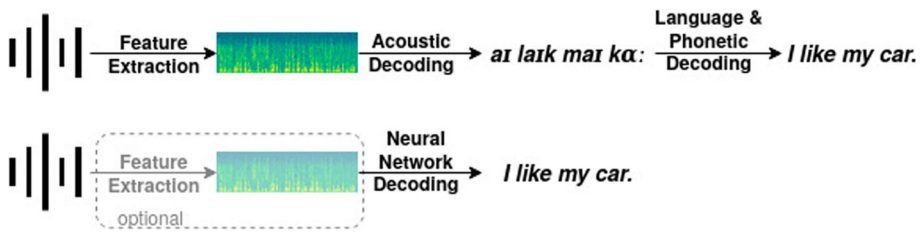

Fig. 1 Pipeline (top) vs. end-to-end (bottom) ASR. In pipeline ASR, the feature extraction is mandatory and during decoding are obtained different output representations. In end-to-end ASR, the raw waveform is directly transformed into text. An additional language model can be used in both cases for rescoring

possible to build every word from a sequence of units. Words cannot be used as base units because they are not generalizable and trainable as opposed to sub-lexical units, such as phonemes or even sub-phonetic units.

2.1.2 End-to-end ASR

The concept of "end-to-end system" (see the bottom of Fig. 1) has more interpretations. The most common definition assumes that an end-to-end speech recognition system is a single neural network that receives raw audio signal at the input and provides a sequence of words at the output, representing the appropriate transcript [11]. The acoustic, phonetic and language components are jointly trained into a single network [12], removing the need for a hand-crafted pronunciation dictionary. Instead, a traditional system comprises some independent modules that work together as a pipeline, each module performing a specific task. As a summary, the features extractor aims to obtain speech features from the acoustic signal. The acoustic model provides the probabilities for each signal sequence, represented by those speech features, given the corresponding sequence of words. The language model deals with the word sequences to match them in phrases with a logical meaning.

The following aspects describe the differences between end-to-end systems and traditional systems, although they may not necessarily be considered as fundamental differences. Some end-to-end considered neural networks, as the one in Section 3.5, which is introduced in [13], are capable of processing raw audio signal, the feature extraction being integrated into the network [14], but also some hand-crafted features which have been already extracted in a previous step. From the acoustic model point of view, while the traditional systems are modeling phonemes, the end-to-end systems are trained on graphemes (characters) [15], word-pieces [16] or even entire words [17]. The output in traditional systems consists in probability distributions over the phonetic units, while the end-to-end systems can do more than that, their output consisting directly in characters. The language model is a standalone component in traditional systems, while in end-to-end systems it is possible to be the same, plugged-in as an external component, but also embedded in the network, as a part of the network performing the task of language modeling. There are also lexicon-free approaches in end-to-end networks, which remove the constraints imposed by a finite lexicon, being possible to handle out-of-vocabulary (OOV) words. This feature must be balanced with another aspect, the risk of obtaining meaningless or misspelled words.

Besides the unitary, homogeneous architecture of the end-to-end systems, fundamental to them is that the training is done from scratch, without using pre-aligned data. In this way, the possibility of using potentially incorrect alignments as training targets is eliminated [18]. End-to-end systems are fully discriminative, everything is learned from data. Standard approaches use an additional generative component, most often this being represented by a Hidden Markov Model (HMM). The hybrid HMM-DNN approaches based on Lattice-Free Maximum Mutual Information (LF-MMI) objective function are also considered end-to-end [19], in the sense that there are not required initial forced-alignments to start training.

2.2 Feature extraction

This subsection describes the most popular speech features used in ASR: spectrograms, Mel-filterbanks, Mel-frequency cepstral coefficients and i-vectors. Feature extraction is a pre-processing technique which transforms the speech waveform in a more compact parametric representation, focusing on some key-properties of the signal, relevant for speech recognition. Regardless the features type, most of them share some main principles, being computed by applying some specific operations. In general, they are biologically-inspired, imitating the human sound perception system. The main operations that are applied in the feature extraction process will be explained below.

2.2.1 Type of features

The speech signal is not stationary, its statistics change over the temporal dimension. Therefore, the analysis is performed on short time frames, in order to respect the quasi-stationary property of speech signal. The first operation is represented by framing, which consists of splitting the signal in short frames, usually 25 ms with 10 ms overlap. The second operation is called windowing and it is done during framing. A signal convolution with a Hamming [20] or Hanning [21] filter is performed, aiming to smooth the frame border discontinuities, in order to avoid the occurrence of frequency artifacts. Because it deamplifies the frame ends, some parts of the signal would matter less. This is why overlapping is essential: it retrieves the information which may be discarded at the border of two consecutive frames. The third most common operation in this pipeline consists in applying the Fast Fourier Transform (FFT) to pass the signal from time domain to frequency domain. As the Eq. 3 shows, the speech signal is composed through the time domain convolution between the base signal, represented by the air exhaled from the lungs, and the time response of the vocal tract. The latter is conclusive for the speech recognition task and the FFT is used to separate it. Therefore, the convolution operation in time domain becomes a multiply operation in frequency domain and the Eq. 4 presents how the logarithm is applied to convert the multiplication operation into a summation operation, which is a linear one and

whose terms can be separated.

$$S(n) = E(n) * h(n) \overset{FFT}{\rightarrow} S(w) = E(w)H(w) \tag{3}$$

$$\begin{aligned} S(w) = E(w)H(w) &\overset{\log}{\rightarrow} \log(S(w)) \\ &= \log(E(w)) + \log(H(w)) \end{aligned} \tag{4}$$

As could be observed in Fig. 2, although the first operations are common to most types of features, there are more or less processed features, depending on the stage at which they were created. The spectrograms represent a very common type of features. They are defined as the power of the frequency bins at a specific time and they are obtained by applying framing, windowing and FFT operations.

The Mel-filterbanks features, named also Mel-Frequency Spectral Coefficients (MFSC), follow the same main steps as spectrograms, being characterized by an additional step that involves applying a bank of triangular Mel filters [22]. Typically, a number of 40 filters is used. They translate the frequencies from a wider initial range to a narrower range. The vocal parameters are grouped around the lower frequencies, where the human auditory system can achieve a much better distinction. Consequently, the lower part of the spectrum contains much more crucial information for speech recognition. In the frequency range 0-1000 Hz the perception is linear, while over this threshold, it becomes logarithmic.

The Mel-Frequency Cepstral Coefficients (MFCCs) [23] are the most commonly used features in speech recognition. Obtaining them involves an additional operation compared to Mel-filterbanks: a final step consisting in the backward transition in the time domain by applying the discrete cosine transform (DCT). Its role is to reduce the dimensionality of the parameters and to achieve their decorrelation, because the filterbanks parameters are highly correlated. From the each frame signal are usually retained the first 13 MFCCs. This value is traditionally used. Because the MFCCs approximate the initial signal, more values make a more accurate, but more computationally complex approach, while fewer values make a thicker approximation. The MFCCs only provide information about the current frame and sometimes context data is needed. To fill this gap, the 1st and 2nd derivatives are computed, known also as delta and delta-delta or differential and acceleration coefficients. They capture information about the signal trends, the variation over the neighboring frames.

I-vectors (identity vectors) [24] are a kind of feature predominantly used for the speaker recognition task, but they have also applicability in speech recognition. They are derived from the Joint Factor Analysis [25] (JFA), where the supervectors have been defined. They consist in the mean components of the Gaussian Mixture Model (GMM) that models the speaker-specific acoustic features. Because i-vectors contain speaker characteristic information, they have a role in improving the system's adaptation to the specific speech of a speaker.

2.3 Traditional, HMM-based acoustic modeling

This subsection presents the key concepts of the HMM-based acoustic modeling, a widespread approach, now considered traditional, with the advent of end-to-end networks. The phoneme is the shortest unit of sound and it usually corresponds to the acoustic manifestation of a letter. Each word can be decomposed into a phoneme sequence. Given the way the words are articulated, the phonemes are context dependent: two identical phonemes will manifest differently if they occur in different contexts with different neighboring phonemes. Consequently, phonemes are usually modelled with three states: the first state models the transition from the previous phoneme, the second state models the central, stationary part of the current phoneme, while the third state models the transition to the next phoneme. A phoneme modelled this way is referred to in the literature [26] with the term *triphoneme*, and each of its states is referred to with the term *senone*. The number of different triphonemes is very large (three to the power of number of phonemes), thus the training process is difficult and intensely computational. However the various senones composing the triphonemes are not totally distinct and, consequently, they can be clustered based on similarity and modeled together.

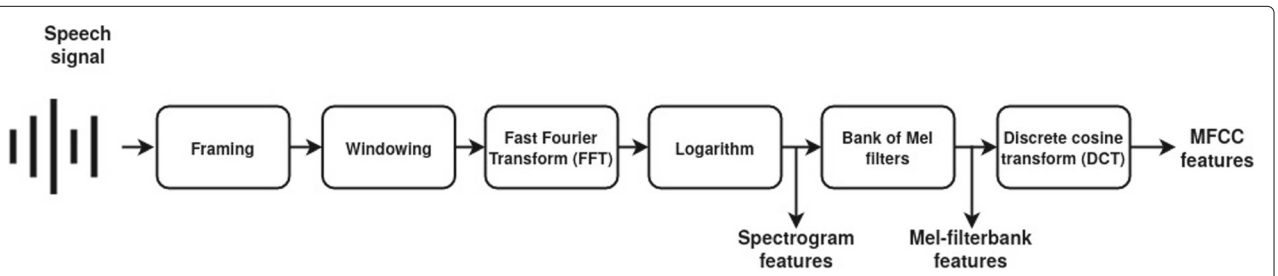

Fig. 2 Different speech features obtained at different processing stages

Traditionally, the feature vectors representing the senones were modeled using Gaussian Mixture Models (GMMs) [27], while the transitions between senones (the actual triphonemes) were modelled using HMMs [28]. Provided the properties of speech, HMMs used in ASR are constrained to have forward-only transitions. HMM-GMM systems for ASR are trained using the Baum-Welch algorithm [29], while the Viterbi algorithm [30] is used for speech decoding.

Taking into account that these states are context-dependent and speech is possible due to transitions between states, it was found that speech can be modeled using the Hidden Markov Models (HMMs). An HMM is a finite state automata where the sequence of states is not known, but we know only the acoustic vectors corresponding to each state, generated by a probability density function. An HMM is characterized by a set of states, emissive or non-emissive, together with a set of transition and loop probabilities. Generally, the HMM transitions have no constraints, except for the ASR systems. Due to the speech sequential characteristic, back off transitions or states skipping transitions are not allowed. Each state may have transitions to itself or to the next state. Each state emits vectors of values based on a probability density function, given by a Gaussian Mixture Model (GMM). The normal probability density function models a range of values, being centered in the highest probability point, corresponding to the mean of the values. It provides the probability that a value belongs to a class. Thus, by choosing an arbitrary point in the value space, we can determine how likely it is to belong to one of the classes. Gaussian Mixtures (GMMs) are weighted sums of Gaussian densities, which try to approximate the probability of acoustic feature vectors, extracted from the signal.

Training a GMM aims to estimate its parameters. This operation is done by using the Expectation-Maximization (EM) algorithm. The first step of the training (E) computes the class belonging probabilities for the input data. The second phase (M) computes the model parameters using the current class belonging probabilities of the input data. The algorithm runs iteratively until convergence to the local maxim of the plausibility function is reached. HMM decoding uses the Viterbi algorithm and it represents the speech recognition task itself. It determines the most likely sequence of acoustic states that has generated a set of speech features.

The first step towards employing deep learning for speech recognition was represented by the replacement of GMMs with DNNs for modeling senones [31]. In this case, the DNN's role was to predict the senone class. While the first attempts used speech corpora annotated at phoneme-level and required no HMM-GMM alignments, the usage of speech corpora annotated at utterance-level was restricted by having phoneme-level alignments produced by intermediary HMM-GMM systems.

The hybrid approach is similar to the traditional HMM-GMM, given that the GMM is substituted by a neural network, using the posterior probabilities output by the last layer. Moreover, the neural network is not trained from scratch, but on top of the alignments provided by an already trained HMM-GMM system. This is because usually the speech corpora are not annotated at phoneme-level. Consequently, the neural network in HMM-DNN approaches is used to predicts the senone class based on the input represented by several frames of feature vectors.

2.3.1 TDNN

Time-Delay Neural Network (TDNN) [32] is a convolutional network which operates on time domain, modeling the temporal dependencies. This is easier to be parallelized than a recurrent network and it is comparable with feed-forward DNN in terms of required training time. The input of each unit in TDNN layers is expanded out spatially in a couple of sequential units from the previous layer. Thus, the lower layers learn a narrow context, while the higher layers process activations for a wider temporal context. The hyper-parameters for a TDNN are represented by the lengths of the input context of each layer. In [33] it is proposed a TDNN which uses a subsampling technique, computing the hidden activations only at some specific time steps. This method avoids redundancy due the fact that the large contexts overlap leads to highly correlated neighboring activations. It was concluded that a larger context on the left side is optimal for online decoding. The model size and the training time are also reduced. Therefore, this subsampling mechanism in TDNN networks is similar to a convolutional operation which allows gaps in the convolutional filter.

Because it was desired to compress the network layers, a factored form of TDNN (TDNN-F) which is derived from Single Value Decomposition (SVD), was introduced in [34] as an improvement over TDNN. It supposes training from a random start and each learned matrix is decomposed as a product of two smaller factors, one of them constrained to be semi-orthogonal. This decomposition is obtained by a linear bottleneck operation. Thus, weights compression is achieved, the smaller singular values are discarded. A matrix M is semi-orthogonal if $MM^T = I$ or $M^T M = I$.

2.3.2 Convolutional + time-delay neural network (CNN-TDNN)

This is another variation of network, based on the purely TDNN approach, before which are added a couple of stacked convolutional layers. Their main purpose is to perform a further processing of the acoustic features, acting as a feature processing front-end. These extra layers

perform temporal convolutions on the speech features, reducing their spectral and temporal variability. The convolutional layers, by their structure given by local connectivity, weight sharing and pooling, have the property to annihilate the small variations that appear in the spectral domain. These variations are induced by both the speaker and the acoustic environment in which the speech takes place [35]. There are some reports [36–39] stating that better results are obtained using CNN-TDNN instead of simple TDNN networks, but the CNN-TDNN requires a little more computing power.

2.4 End-to-end ASR systems

This subsection aims to summarize the most common approaches in end-to-end systems. We provide a brief overview of the neural networks used for acoustic modeling, presenting their main characteristics.

2.4.1 Architectures

End-to-end systems refer to a unitary architecture, capable to receive raw audio or speech features as input and it outputs words. Despite a traditional, pipeline system, all the components as feature extractor, acoustic model and even sometimes the language model are integrated in one single network. Specifically, the loss function of the neural network is set directly at the character level. The following paragraphs include a review of the most popular end-to-end architectures, along with some specific details for each.

Some state-of-the-art acoustic modeling approaches using neural networks are based on simple stacked convolutions [13, 40]. The speech features are received as network input and they are passed over a series of convolutional layers, each one being characterized by the number of filters and their dimensionality. The loss function is directly set at the character level facilitating the output of the networks to be represented by words. Some well-known architectures based on stacked convolutions are those presented in [13, 40].

In [41, 42] are presented convolutional architectures characterized by dense residual connections. The residual connections work as a bypass, connecting early layers to later layers. The term *dense* denotes connections between each layer to every other layer, similar to a fully-connected network. In the dense residual networks, all the outputs from the preceding layers are concatenated in order to provide the input for the subsequent layers. The total number of connections is $L(L+1)/2$, where L represents the number of layers, unlike a regular network where the number of connections is just L, one between each two consecutive layers. Among the advantages of this approach are the avoidance of vanishing gradient in very deep networks, better feature propagation and network parameters reduction.

The recurrent approaches succeed in modeling long-range dependencies over temporal sequences. In [43] are described some variations of recurrent architectures used for acoustic modeling. The recurrent neural networks (RNN) are characterized by cyclic connections, the output from the previous step feeds the input at the current time step. The Long Short Term Memory (LSTM) networks are more complex RNN networks. They contain memory blocks, that in turn contains cells with self-connections, aiming to conserve the temporal state. Besides the cells, there are some gating mechanisms, like the input, output and forget gates, which control the dataflow. Some LSTMs also have peephole connections [44] between the internal cells and gates. One of the advantages is that the internal state of a cell could be inspected even if the output gate is closed. Another one is represented by the ability to learn precise and robust timing intervals between relevant signal events. The bidirectional LSTMs (BLTSMs) are networks that process the input in both directions: from the past to the future and vice versa. It was shown that a two layer LSTM outperforms a DNN with an order of magnitude more parameters. The DNN drawback consists into the limited temporal modeling. DeepSpeech [45] is a well-known recurrent-based ASR framework which obtains competitive results with just few bidirectional layers.

The encoder-decoder architecture [46–48] is a neural network specialized in sequence to sequence mapping. Besides its application in speech recognition, is was initially applied in machine translation and language modeling. Usually, the encoder and the decoder are implemented by recurrent networks, thanks to their ability to work with time-dependent sequences of data. Both of them are jointly trained. Basically, as it is described in [46], the main idea revolves around a sequence of input data which is passed to the encoder where the fixed-length context vector is composed. The decoder is auto-regressive, it consumes the previously decoded symbols as well as the context vector in order to predict the following symbol.

Attention comes as an improvement of the encoder-decoder architecture. It aims to bypass the limitation of fixed-length encoding vector. Working with long sequences could be an issue because the neural network must be able to compress all the information in one vector. In [49] it is shown that the performance of an encoder-decoder network decreases as the length of the input increases. The attention approach encodes the input into a sequence of vectors and chooses a subset of them during the decoding. The decoder performs two operations simultaneously: aligning and decoding. Each time the network tries to generate a new output symbol, it looks at a set of positions in the source sequence where the relevant information is concentrated. The prediction is computed considering the encoding vectors for these positions as

well as the previous predicted symbols. Thus, instead of having one single encoding for the whole sequence, there is a corresponding context vector for each output symbol. This is computed as a weighted sum of the encoder hidden states, where each hidden state h_i carries on information about the whole sequence, but focusing on the input frames around the *i-th* position.

Another encoder-decoder architecture is presented in [50]. This is an attention-based encoder-decoder, where the novelty is represented by the encoder, a fully convolutional one, composed by time-depth separable (TDS) blocks. A TDS block comprises two parts: a 2D convolution layer, both in time and features space domain, followed by two convolutional layers with 1x1 filter, which work as a fully-connected feed-forward layer. This approach has the ability to generalize much better than other convolutional networks, working with a reduced number of parameters, while the receptive field is kept large. The decoder is a conventional recurrent-based one, but having a particularity regarding the training step: the previous ground-truth is used instead of the real previous prediction. This technique is called teacher forcing. The idea behind aims to discard recurrent dependency mechanisms in order to facilitate parallel computing.

2.4.2 Loss functions

Conventional DNN acoustic models are using a frame-level objective function to perform training based on the alignments provided by a HMM-GMM system, commonly the cross-entropy (CE) function. The cross-entropy objective function is described in Eq. 5, where t iterates over THE AUDIO FILES in the training set, i over the time frames in each audio file and $\hat{y}_i$ represent the labels for each frame provided by forced-alignment.

$$F_{CE}(\lambda) = -\sum_{t=1}^{T}\sum_{i=1}^{I}\hat{y}_i(t) log\ y_i(t) \tag{5}$$

Connectionist Temporal Classification (CTC) [51] allows to train a network without being required a frame-level alignment between the speech signal and the transcripts from the training dataset. Standard ASR systems use a statistic (e.g. GMM) or deep learning (e.g. DNN) component to predict what is being uttered and a time consistency component(e.g. HMM or CTC) to handle the context, the previous and the future frames. The CTC approach implies a softmax component as a final layer of the network, in order to provide a distribution of probability over all the possible output symbols. The output could be visualized as a $G_{CTC}(\theta; T)$ graph for a given transcription θ over T time frames. Each time frame is defined by a set of nodes and each node representing a possible output label, given by a probability distribution function $f_t()$. A path $\pi = \pi_1, ..., \pi_n \in G_{CTC}(\theta; T)$ represents a potential transcription through this graph. A sequence level objective function, derived from maximum likelihood, operates on this distribution aiming to maximize the probability of the correct symbol:

$$CTC(\theta, T) = -logadd\sum_{t=1}^{T} f_{\pi_t(x))}, \tag{6}$$

where $logadd()$ is an exponential function applied on a sum of logarithms, working as an improved version of $max()$. Specific to the CTC is the use of an additional blank symbol (-), which is required to model the non-phoneme emissions. Given the usual terminology, the task of determining the labels is called decoding. The trivial approach is the max-decoding or greedy decoding, which supposes to take the most probable phoneme at each time step. Because the predicted sequences have variable length, it is possible for a phoneme to be successively predicted. The blank label is used to separate the legit repetitive characters by those that are repeated due the variable speech rate or to mark the transition from one character to another. In fact, to obtain the final transcript, repeating characters are collapsed and the blank labels are discarded. A CTC drawback is represented by the assumption that the output label at a given moment of time is independent by the previous output labels. This fact is imposed by the architecture, there is no feedback loop from the CTC output layer to itself or to the network. However, CTC decoding can be integrated with a language model, so that the transcription of non-existent words is avoided. Another approach presented in [52] consists in a loss function as a joint between CTC and attention mechanism.

Lattice-Free Maximum Mutual Information (LF-MMI) objective function is used in chain models [53] to perform sequence-discriminative training. The traditional MMI aims to maximize the posterior probability and it is as follows:

$$\begin{aligned} F_{MMI}(\lambda) &= \sum_{r=1}^{R} logP_\lambda(S_r|O_r) \\ &= \sum_{r=1}^{R} log\frac{P_\lambda(O_r|S_r)^k}{\sum(O_r|S)^k P(S)^k}, \end{aligned} \tag{7}$$

where S_r is the correct transcription of the r^{th} speech file O_r, $P(s)$ is the language model probability for sentence s. The numerator provides the likelihood of data given correct word sequence (reference alignment), while the denominator provides the total likelihood of the data given all possible word sequences, being equivalent to summing all possible word sequences estimated by the full acoustic and language models. The numerator encodes the supervision information and it is specific for each utterance, while the denominator encodes all possible

word sequences and it is identical for all the utterances. This objective function is optimized by maximizing the numerator and minimizing the denominator.

Therefore, the chain models are trained from scratch, without any prealigned data, trying to offer an alternative solution for CTC and attention mechanisms.The authors in [53] reported that their efforts to get CTC to beat cross-entropy were unsuccessful, but some ideas from CTC could be used in the sequence-discriminative LF-MMI criterion. CTC and MMI maximizes the conditional log-likelihood of the correct transcript, but the probabilities in CTC are locally normalized, while in MMI are globally normalized. LF-MMI supposes to create the denominator using a phone-level language model instead of word level one. This language model is estimated from phone-level alignments of the training data. Another characteristic of the chain models is the 3-times smaller output frame rate, allowing the HMM to be traversable in one transition, instead of three. The real-time decoding becomes faster as well.

The Auto Segmentation Criterion (ASG) [13] criterion was developed as an improvement over CTC. It introduces a dependency between the output symbols, by using transition probabilities between them. Moreover, there are a couple of new features of ASG. The output graph is less complex because there are no blank labels. I was empirically found that there is no advantage when blank labels are used to model the garbage [13]. Instead of it, the repetition of the output symbols is modeled by a repetition character label. Another characteristic is given by the un-normalized scores of the nodes. It facilitates external LM plug-in, which would provide transition scores between nodes. Also, specific to the ASG is global normalization instead of per-frame normalization, leading to low confidence for incorrect transcriptions. Therefore, the score of a given sequence of words W is given by:

$$ASG(W) = logadd \sum_{t=1}^{T} f_{\pi_t}^t + g_{\pi_{t-1},\pi_t}, \tag{8}$$

where $f()$ denotes the probability of an output symbol at the t time step and $g()$ is the transition probability between two consecutive symbols.

2.5 Language modeling

In this subsection we describe the language modeling approaches used in the systems we are going to analyze later. As explained in Section 2.1, the language model is one of the components of an ASR system, which can be integrated as a distinct component, and in the end-to-end systems its role can even be performed by a part of the neural network.

The role of the language model is to estimate the likelihood of a word sequence $W = w_1, ..., w_n$ to form a valid sentence. A language model is useful to take decisions when the acoustic model output is composed by a set of phonemes which could form multiple alternative sentences. Even these alternatives are very similar from the acoustic point of view, the LM will choose the one that makes more sense.

2.5.1 N-gram/Probabilistical LM

The n-gram model provides a statistical view over how words are combined to form valid sentences. It assumes that a word depends only on a fixed number of previous words. The probability of a sequence of words is considered as a set of probabilities, where the probability of a given word depends by the preceding words:

$$\begin{aligned} p(w) &= p(w_1, w_2, ..., w_n) \\ &= p(w_1) * p(w_2|w_1) * p(w_n|w_1, w_2, ..., w_{n-1}). \end{aligned} \tag{9}$$

Words occurrence or succession probabilities could be computed by taking into account a large volume of text. The most common n-gram models are 2-gram and 3-gram, where a history of one or two words respectively is required. For instance, the probability of a pair of words for a 2-gram model is computed as:

$$p(w_j|w_i) = \frac{count(w_i, w_j)}{\sum count(w_i, w)}. \tag{10}$$

The occurrence probability of the words pair (w_i, w_j) is given by the number of occurrences of the word w_i followed by the word w_j, divided by the number of occurrences of the same word w_i, followed by other words.

2.5.2 Neural network based language models

Instead of n-gram models or a feed-forward DNN network which learns from a fixed context, RNNs are capable to learn also from all the previous words. The recurrent neural network based language model presented in [54] is a simple one. It comprises one input layer, one hidden layer and one output layer. The input at current time step consists in the current word, w, and the hidden state from the previous time step. The output layer represents the probability distribution of the next word given the previous word the context. Because these algorithms cannot directly work on text and label encoding (using integers) could be confusing for the network, inducing the idea of order or hierarchy, they are using one-hot-encoding (1-of-k) for words, where k is the number of words from vocabulary. One-hot-encoding associates a binary vector to each word in vocabulary.

RNNs are difficult to train using backpropagation due the vanishing gradient problem. The gradient propagated back through the network decays or grows exponentially as context get longer. LSTMs provide an alternative to avoid this issue, using a different memory cell, while the rest of the algorithm remains unchanged, being similar

with the RNNs. The LSTMs take as input the previous hidden state and the current input. The cells decide what to preserve and what to remove from the memory. In [55] is presented a topology which consists in an input layer, two hidden layers, where the first one is a projection layer and the second layer is a recurrent one, using LSTM cells. The projection layer performs the projection of all words in the context in a continuous space.

A new type of convolutional network [56], based on gated linear units (GLU), is able to outperform LSTMs for the language modeling task, both in terms of accuracy and implementation, being easier to parallelize and less complex. This approach performs a convolution operation over the input aiming to remove the temporal dependencies.

The Transformer-XL [57] network can learn dependencies without constrains regarding the fixed-length context. It captures longer dependencies than simple Transformers or than RNNs, achieving better performances on short and long sequences and a faster inference time. Its particularity consists in reusing the hidden states from the previous segments, instead of computing them each time for each new segment. These reused states play the role of a buffer memory for the current segment, linking the segments in this way and propagating the information over longer contexts.

2.5.3 Language model integration

In shallow fusion [58, 59], the AM proposes at each time step a set of possible phones, which are scored by a weighted sum of scores given by the AM and the LM. The shallow fusion formula is given by:

$$\hat{y} = argmax\ log(y|x) + \lambda p_{LM}(y), \tag{11}$$

where the first term is the AM probability and the second one is the LM probability. This fusion takes place at the inference time.

Deep fusion [58] is based on concatenation of acoustic model and language model hidden states next to each other. Both models are trained separately and their fusion is made using a gating mechanism. The output probability of the next word is given by this model which is fine tuned to use both of the hidden states. The hidden layer of the deep output takes as input the hidden state of the LM in addition to that of the AM. The biggest disadvantage with deep fusion is that the AM and the LM are trained independently. This fact could be an issue in encoder-decoder models, because the decoder is learning a language model from the training data labels, which can be poor compared to the large text corpora used for LM training. The decoder must overcome this limitation, being able to incorporate the new language information. Another issue occurs if the AM and LM are trained on different domain corpora. If they are deep fused, the decoder will tend to follow the linguistic style learned by the AM.

Cold fusion [60] concept is derived from the deep fusion, the main difference lies in fact that the end-to-end acoustic model is trained from scratch together with a pre-trained LM. During the training process, the AM learns to use the relevant information from the LM to correctly map the source sequence to the target sequence. If there are uncertainties at the decoding step caused by the AM (noisy speech, out-of-vocabulary words), the fusion model learns to take advantage of the LM. Cold fusion uses a different gate for each hidden node of the LM. This improvement allows the decoder to choose which information given by LM fits better at a specific time step. In [60] was shown that a decoder using cold fusion outperforms a pure end-to-end attention based system, even if the last one is using $4x$ number of parameters. Also, the training time is speed-up when cold fusion is used. Domain transfer is easier when cold fusion is used. Only a small amount of labeled data is required to close the gap between domains.

In [61] is proposed a novel LM integration approach, where a pretrained LM should represent a lower layer of the decoder of an attention-based encoder decoder system. Thus, more tight word embeddings to the context are provided.

2.5.4 Rescoring

The output of an ASR is not a simple word sequence hypothesis corresponding to the acoustic signal, being more advantageous to keep more information. This fact is done by generating a *lattice*, a graph $G(N, A)$, where N represents the nodes and A represents the arches. The output of the decoding could be structured as a lattice, where each arch has a specific probability and a path through the graph is an alternative transcription. The path with the best probability leads to the best transcription hypothesis. Lattice rescoring [62, 63] implies processing all the probabilities and replacing them with new ones provided by a better language model. The difference between rescoring and shallow fusion is as follows: the rescoring operation performs over the n-best hypotheses produced after the beam search, while the shallow fusion performs a log-linear interpolation between AM and LM score after each beam search time step.

3 State-of-the-art ASR implementations

The various ASR system architectures presented in the previous sections differ from many points of view. They comprise various types of acoustic and language models, some are based on a multi-component pipeline structure, while others are end-to-end neural networks, etc. This leads to systems with fundamentally different

complexities, in terms of model size (or number of parameters) and activations, which influence directly the memory load, and in terms of number of operations performed for transcribing speech, which influences directly the real-time factor. These are crucial performance figures, which one must take into account, along with the transcription quality (measured in word error rate), when choosing the architecture to be implemented and deployed in embedded applications. Consequently, this section is dedicated to the comparison of the most popular, modern ASR implementations in terms of a trade-off between system complexity and accuracy. To the best of our knowledge, this is the first comparison of such scale created for modern automatic speech recognition systems.

The various ASR systems evaluated and compared in this section are the following:

- Kaldi's pure-TDNN [64] - a lightweight, multi-component ASR system that uses a time-delay neural network for acoustic modeling and an HMM for sequence modeling;
- Kaldi's CNN-TDNN [65] - an extension of the previous system that processes the input features with 1-D convolutional layers;
- DeepSpeech2 implementation from PaddlePaddle [66] - an end-to-end bi-directional RNN with convolutional layers for speech feature processing;
- RETURNN from RWTH [67] - an attention-based encoder-decoder that outputs word parts;
- Facebook CNN-ASG [68] - a fully convolutional end-to-end network that uses a CTC-derived criteria, ASG, being able to output characters;
- Facebook TDS-S2S [69] - an end-to-end encoder-decoder with time-depth separable convolutions, trained with sequence-to-sequence (S2S) attention mechanism;
- Jasper from Nvidia [70] - an end-to-end deep neural network based on time-delay convolutional interleaved with fully connected layers and characterized by residual connections;
- QuartzNet from Nvidia [71] - a Jasper derived end-to-end deep neural network based on 1D time-channel separable convolutions.

The systems mentioned above are analyzed and compared from a structural point of view, providing information regarding the network input and output type and dimension or the type, number and size of the component layers. Based on these values, we compare the networks in terms of number of parameters, operations and activations, thus offering insights into how they could be implemented and deployed in embedded applications and with what costs.

A fair comparison of the system complexity vs. accuracy trade-off can only be made in the context of a specific speech recognition task, because the ASR systems are usually adapted (number and size of layers, size of vocabulary, etc.) to each task. The most popular speech recognition tasks/ corpora are presented in Table 3. Benchmarking for English speech recognition is usually performed on one of these tasks. The table shows the ASR frameworks that are the subject of our comparison, all providing adapted ASR systems for LibriSpeech [72], while only three of them also provide adapted systems for Wall Street Journal (WSJ) [73]. In this context, we decided to focus the analysis on the LibriSpeech case study.

Librispeech [72] is one of the most popular freely available English dataset, presenting a great variety of data, both through the large number of speakers and the number of hours composing this speech corpus. It contains 1000 hours of read speech from public domain audio books, provided by approximately 2400 speakers. This is a widespread task, most of the well-known ASR frameworks contain adapted systems for it, being possible to compare them in terms of WER.

3.1 Kaldi chain model TDNN

This model is part of an implementation for the LibriSpeech task existing in the Kaldi toolkit [64]. It corresponds to a multi-component system, consisting of a TDNN based acoustic model, a phonetic model and a language model, all these being the core components of the pipeline system. This system is a hybrid one, the acoustic model consists of a TDNN which jointly work with an HMM, as presented in Section 2.3. The TDNN network

Table 3 Comparison of the most popular speech datasets used for ASR evaluation

ASR task	Speech type	Size [h]	# of speakers	Framework				
				K	P	W	R	N
LibriSpeech [72]	read speech	960	~2400	✓	✓	✓	✓	✓
WSJ [73]		80	284	✓		✓	✓	
TED-LIUM2 [74]	TED talks	207	1242	✓			✓	
Switchboard [75]	conversational telephone speech	300	543	✓			✓	
Fisher [76]		2742	~12400	✓				

We compare the type of speech and dataset size, expressed in number of hours of speech and number of speakers. The recipes available in various ASR frameworks: K - Kaldi; P - PaddlePaddle DeepSpeech; W - Wav2Letter; R - RWTH Returnn; N - Nvidia (OpenSeq2Seq & NeMo)

outputs the probability that an acoustic signal part corresponds to a subphonetic unit. The HMM manages how these units can be linked together. The phonetic model functions as a lookup table that determines to which word a sequence of phonemes corresponds, while the language model estimates the likelihood of a sequence of words. Optionally, a more complex language model for rescoring can be used, which improves the initial transcription. Typically, the language model used for decoding is a probabilistic n-gram of order 2 or 3, while the rescoring operation uses a more complex, higher-order n-gram (Section 2.5.1) or a RNN-trained model (Section 2.5.2).

Two types of features are used as the input of the TDNN network: 40-dimensional high-resolution MFCCs extracted from frames of 25 ms length and 10 ms shift and 100-dimensional i-vectors computed from chunks of 150 consecutive frames, equivalent to 1.5 seconds of speech. Three consecutive MFCC vectors and the i-vector corresponding to a chunk are concatenated, obtaining a 220-dimensional feature vector for a frame. The components are decorrelated by applying Linear Discriminant Analysis (LDA), without changing the dimensionality of the data. Therefore, the network input is a 220-dimensional feature vector (Feature type #1). More details about these features are illustrated on the left side of Fig. 3.

The network trunk consists of a cascade of 16 factored time-delay blocks (TDNN-F), preceded by a simple TDNN block. As it was explained in the last paragraph from Section 2.3.1, there is a main difference between a TDNN-F block and a TDNN block; the TDDN-F block comprises a linear-affine sequence of operations that act like a bottleneck transforming the 1536-dimensional input vector into an 160-dimensional intermediary vector and then back into an 1536-dimensional output vector. This is based on the matrix decomposition technique and it is useful for parameter compression. Particular to this implementation, the TDDN-F block ends with a summation operation that adds the output of the current processing block to the down-scaled (75%) output of the previous block: this acts like a residual connection. Therefore, the TDNN performs 1-D temporal convolution, applying the operations on the current input vector as well as some previous and some future input vectors. The contexts differ from one block to another. The TDNN Blocks 2-4 process the input vectors at time indexes t-1, t, t+1. The TDNN Block 5 processes only the input vector at time t. The TDNN Blocks 6-17 process the input vectors at time indexes t-3, t, t+3. Those blocks are using the subsampling technique: some time-frames are ignored during the temporal convolutions, the network having in this way a larger receptive field. An overview of the network is depicted in the central part of Fig. 3.

It has been empirically proven [77] that the network performs better if it has two output blocks. The first one is based on cross-entropy, called *xent* in Kaldi. The other one is based on the chain loss function, which uses the LF-MMI criteria. Both of them are explained in Section 2.4.2. Each one is composed by affine, Rectified Linear Unit (ReLU), batch normalization and again the affine layers. They differ by the log-softmax operation applied at the end of the cross-entropy based block. The training process is using both blocks, while the inference only uses the chain based block, because chain models are trained with sequence objective function. The output of the network is 6016-dimensional and it consists in posterior probabilities for the acoustic states, while the output at the entire system level is given by the size of the vocabulary of the language model, equal to 200k words in our case. The output blocks are presented in the left bottom part of Fig. 3.

3.2 Kaldi chain model CNN-TDNN

This model [65] represent a variation of the previous simple TDNN model, being also implemented in the Kaldi toolkit as an approach for the LibriSpeech task. It is part of a multi-component system, which comprises an acoustic model, a hybrid one based on TDNN-HMM, a phonetic model and an n-gram language model. In a similar way, a more complex n-gram or neural based language model can be optionally used for rescoring.

In terms of network input features, Mel-filterbanks are used in this implementation instead of MFCCs. The final features are organized as a matrix, unlike the previous case of simple TDNN where the input features are represented as a vector. Therefore, the input in the CNN-TDNN network is composed of two types of features: 40-dimensional Mel-filterbanks extracted from frames of 25 ms length and 10 ms shift and 200-dimensional i-vectors computed from chunks of 150 consecutive frames. The 40 components of the current Mel-filterbank vector and the 200 components of the chunk's i-vector are organized in a 40x6 matrix of speech features (Feature type #2). The feature extraction procedure and the way they are organized are illustrated in the central part of the left column of Fig. 3.

The neural network component of the acoustic model is very similar to the previous Kaldi TDNN network, the main difference is represented by a few CNN layers placed before the time-delay layers, which act like a front-end block. Three matrices of speech features (Feature Type #2) are provided as input for the Conv. Block 1: the features for the current, previous and next acoustic frames, or, equivalently, a feature volume of 6 x 40 x 3. It uses 64 filters of size 3x3 to perform time and feature space convolutions and outputs a 64 x 40 x 1 volume.

The CNN blocks with front-end role are followed by 12 blocks of factored TDNN (TDNN-F). The first TDNN-F (TDNN-F Block 1) processes only the current time frame, while the rest of them are performing temporal

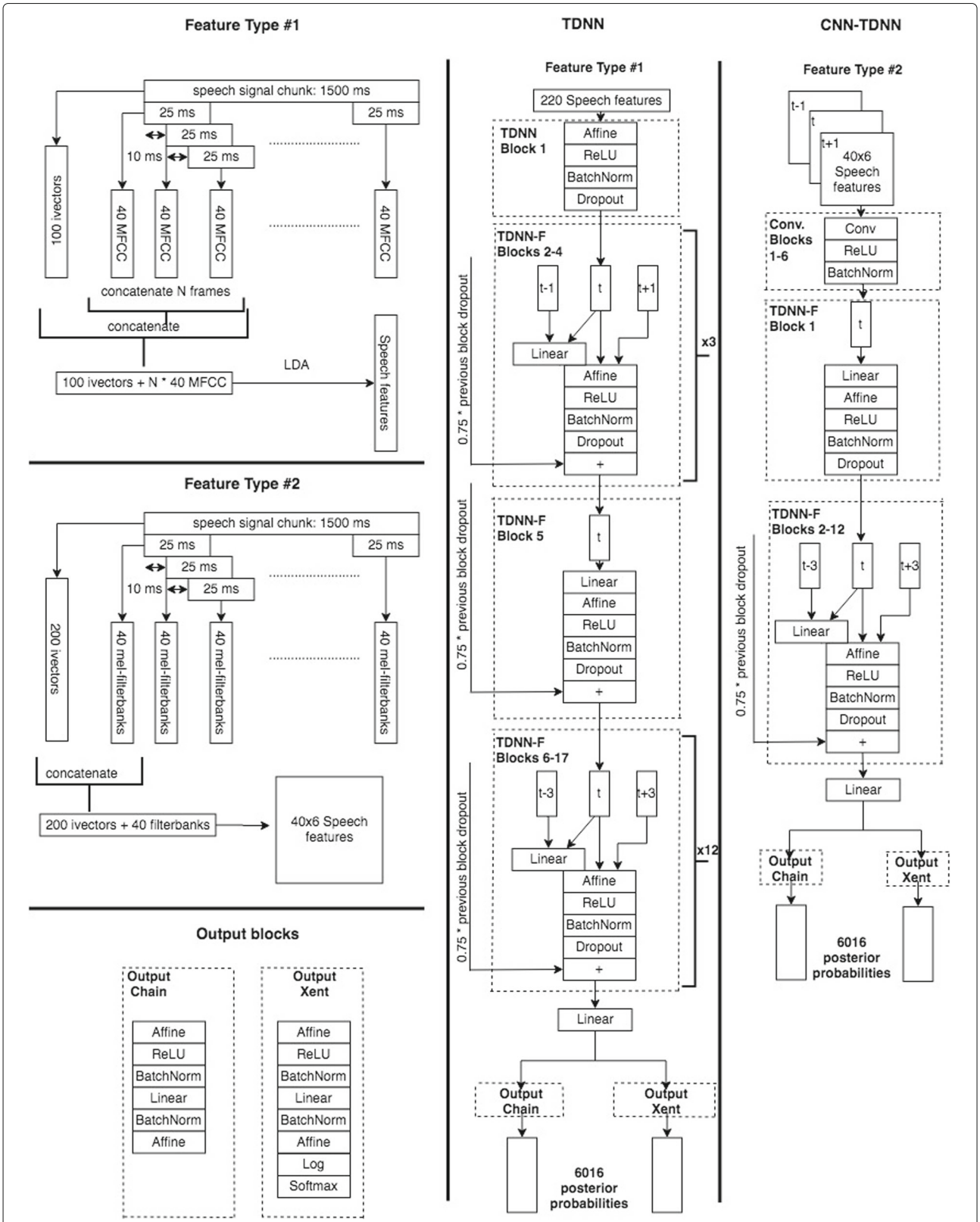

Fig. 3 Feature types and architectures used in Kaldi-based ASR systems. Feature type #1 (upper left) is used in TDNN models (center) and feature type #2 (middle left) is used in CNN-TDNN models (right). The output blocks (bottom left) that use cross-entropy and chain loss functions are used in both architectures

convolution over the time index t-3, t and t+3. The input vectors at time indexes t-3 and t are spliced together into the linear layer, while the input vectors at time indexes t and t+3 are spliced together into the affine layer. The entire CNN-TDNN network is depicted on the central column in Fig. 3.

The output blocks of the CNN-TDNN are identical to those from the simple TDNN architecture. The neural network output is represented by 6016-dimensional posterior probabilities of the acoustic states, while the output of the system is given by the 200k words language model.

The input of the Conv. Block 2 consists of three time-consecutive volumes as the one output by Conv. Block 1, which are spliced together to form the 64 x 40 x 3 feature volume. The second convolutional block applies another 64 filters of size 3x3 to perform time and feature space convolutions and outputs a 64 x 40 x 1 volume. More filters are applied in Conv. Blocks 3 – 6, from 128 up to 256, while the size of the feature volume is kept constant by decreasing the height from 40 to 20 and finally to 10. The convolutional blocks are providing a 2560-dimensional output which is passed to the succeeding time-delay blocks.

3.3 Paddle Paddle implementation of DeepSpeech2

This model [66] represents an implementation of the DeepSpeech2 [45] algorithm created by PaddlePaddle (PArallel Distributed Deep LEarning) to address the LibriSpeech ASR task. This is an end-to-end (E2E) system composed by a single neural network (see Fig. 4) which processes audio features and provides words at the output, as described in Section 2.4. There is no need of a phonetic model and the language model is optional, but it brings transcription improvements, limiting the occurrence of non-existent words. This system allows the integration with a probabilistic n-gram language model by the shallow fusion method, as explained in Section 2.5.3.

The feature extraction step takes place in a previous step, outside of the neural network. The signal is windowed and 160-dimensional spectrograms are computed from a frame of 20 ms length and 10 ms overlap. The processed chunk has the length equal to 160 frames, 1.6 seconds, corresponding to the time sequence processed at once by the network.

The network is considered to be recurrent based as described in Section 2.4.1, but the first layers are convolutional layers, which have more of a preprocessing role of the signal. Therefore, the first two layers perform both time and features space convolution. The first layer applies 32 filters of size 41 x 11, with a stride equal to 3 and 2, respectively a padding equal to 20 and 5, over the 160 x 160 input dimension. The second layer receives the 54 x 81 x 32 output of the previous layer and it performs also a convolution operation using 32 filters of size 21 x 11. A stride equal to 1 and 2, respectively a padding of 10 and 5 are used. This layer outputs a volume of 54 x 41 x 32.

The following 3 layers are all bidirectional recurrent layers. The sequence length is 41, corresponding to the feature dimensionality after the convolutional transforms. The input size of the first recurrent layer is 3776, being equal to the time length after the convolution, 54, by the number of channels, 32, adding the RNN layer size, 2048. The input for the other two recurrent layers in 4096, as the sum of the size of the current RNN layer and the size of the previous RNN layer. A batch normalization operation is performed after each layer. The network is trained using the CTC loss function, explained in detail in Section 2.4.2. This is more than a regular loss function, because it consists of a distribution probability over the output symbols, but it also manages how the symbols succeed, from this point of view having a similar role to HMM.

The output of the network is 30-dimensional, representing the characters set. If the language model is plugged-in, the output size of the system becomes 200k and it consists of the words existing in the model.

3.4 RWTH RETURNN

This model [67] is one of the RWTH RETURNN implementations for the LibriSpeech task. The system, presented in detail in [78], is an end-to-end one consisting in an attention-based encoder-decoder neural network architecture with recurrent layers. The network gets as input hand-crafted features and outputs subword parts, created via byte-pair encoding (BPE) [79]. The final output is given by the language model, an n-gram or an LSTM-based (Section 2.5.2), both of them can be integrated by shallow fusion (Section 2.5.3).

The input is computed on-the-fly, 40-dimensional MFCC features are extracted using a window of 25 ms with 10 ms shift, over a sequence of 2 seconds.

The encoder is composed by 6 stacked bidirectional LSTM layers, having the hidden size equal to 1024. The input of the first layer is represented by the extracted features, while for the other layers the input is 2048-dimensional, as the concatenation of forward and backward of the previous layer. After the forward and backward sublayers, the dropout is applied. The sequence length decreases due to the pooling operation. Therefore, the initial sequence length is 200, halving after the layers with the index 0, 1 and 2.

The output of the LSTM later serves as input for 3 entities: the encoder context, the inverse fertility factor and the CTC mechanism. The encoder context represent the encoder state, the concatenation of the forward and backward hidden states from the 6^{th} LSTM layer, on which was applied a pooling operation to reduce their size to 1024.

The CTC is used as an additional loss function, in order to help the convergence. Using some recurrent links,

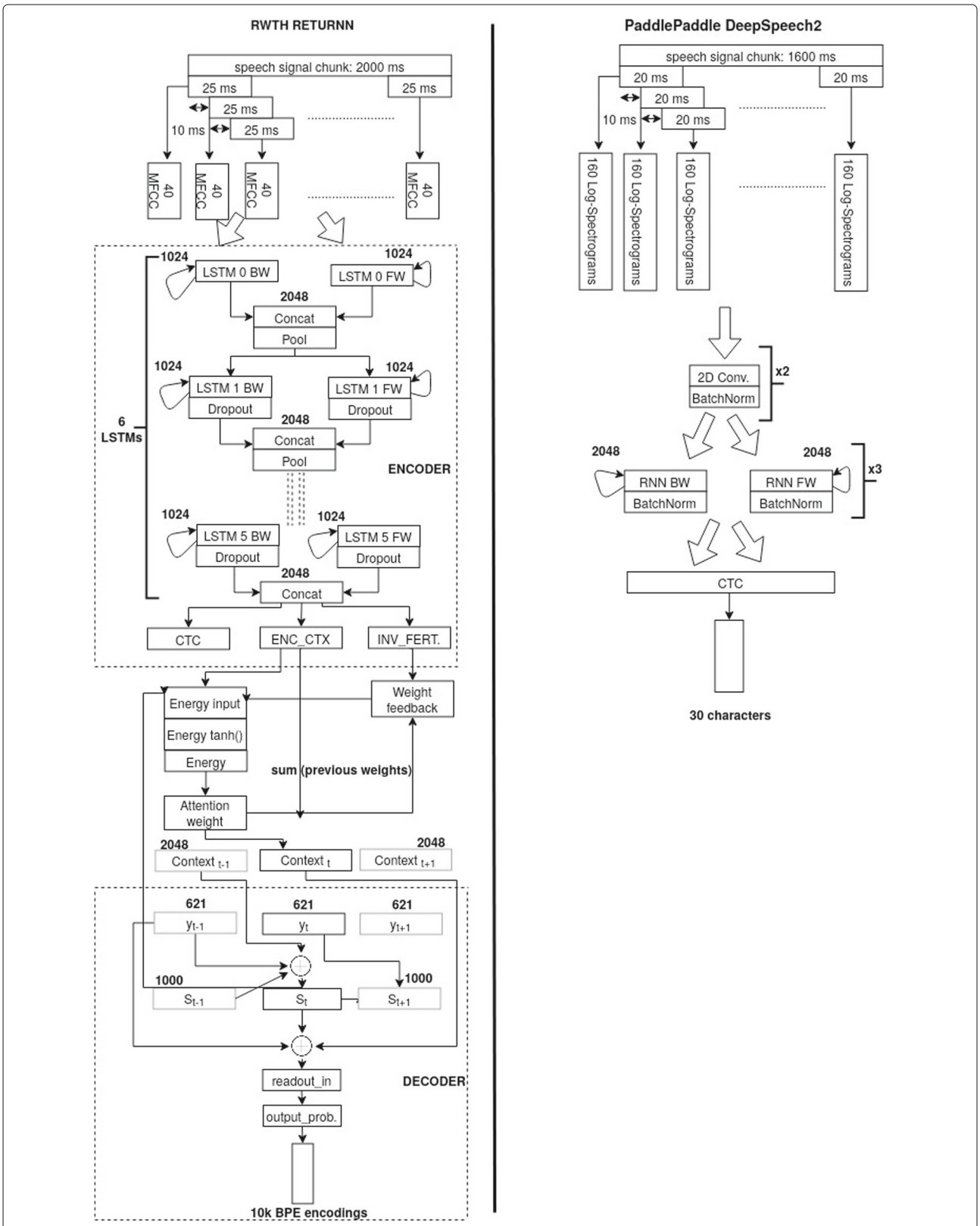

Fig. 4 RWTH RETURNN (left) and PaddlePaddle DeepSpeech2 (right)

which take over the previous output embedding of the decoder (y_t), as well as its hidden states (S_t), the weight feedback and the energy factors are computed. Based on these, the attention feedback factor is obtained, which controls the influence of each state of the encoder in obtaining each state of the decoder. The *readout_in* element takes as input the output embedding, the decoder hidden states as well as the encoder context weighted using the attention mechanism. The result is passed to the *output_prob* element, which provides a probability over the final network output, represented by 10026 subword parts.

The probabilistic model we used is the 4-gram with 200k words. Another language model we tried is a two layer LSTM network, integrated by shallow fusion as a subnetwork at the inference time. A detailed overview of the whole system is illustrated in the left side of Fig. 4.

3.5 Facebook CNN-ASG

This model [68] can be found in Wav2Letter toolkit from Facebook, being specially created to address the LibriSpeech task. The system around it is considered to be end-to-end and depending on the recipe, it may vary from the following points of view:

- it can get as input raw audio [80], power spectrum, MFCCs or Mel-filterbanks;
- it can use a lexicon or it can be lexicon-free [81]; the lexicon acts like a phonetic model, it consists of a mapping from words to their representation as a sequence of tokens, where the tokens are the acoustic units;
- the system outputs a score over the acoustic units, which consist in phonemes, graphemes or word pieces;
- as the output is represented by characters, the system may work without a language model, or it can be plugged-in by shallow fusion an n-gram model or a neural network language model [56], as presented in Section 2.5.2.

We will refer to the fully convolutional recipe (Conv. GLU) [68] where the neural network takes as input Mel-filterbanks, called also Mel-Frequency Spectral Coefficients (MFSC), described in Section 2.2.1. The output of the network consists in scores over the characters set. The system uses a lexicon and an n-gram language model. Another recipe that is a bit different from the system point of view, but uses a similar neural network is the lexicon-free one [82].

Regarding the input of the system, this framework computes features on the fly, prior to running over the neural network. The input of this network is represented by 40-dimensional Mel-filterbank features, extracted from audio frames of 25 ms length and 10 ms shift, processing at once a sequence of 240 frames, which means 2.4 seconds.

The architecture of this network is fully convolutional, as explained in Section 2.4.1. This is composed by 17 1D, time-convolution blocks, each one being characterized by a weight normalization operation [83], the convolution itself, whose output is passed to a Gated Linear Unit (GLU) [56] and finally, the dropout technique is applied. The filter size increases with a unit from one layer to another, the first value being 13 and the last 29. The stride value is always equal to 1. The padding value is also equal to 1, excepting the first layer, which has a padding equal to 170. The number of the output channels increases from one layer to another, the first value is 400, while the last is 1816, where each value is with 10% greater than the previous. The number of input channels is equal to 40 for the first block, the following input channels being equal to half of the number of the previous block output channels, due to the GLU dimensionality reduction. The GLU performs a element-wise product of the first half of its input and the other half, after it was passed through the sigmoid function. After the convolutional blocks, the next layer is a reorder layer, thus the number of input channels becomes the number of output channels, being equal to 908.

The output layers are two final linear layers, on which is applied weight normalization, as well the GLU and the dropout mechanism for the second last. They transform the number of input channels from 908 to 1816 output channels, respectively from 908 to 30, which is the output size of the network, the number of classes, where each class correspond to a character. The final system's output is 200k words, as the number of unique words from the n-gram language model. Therefore, the network is trained to output letters, this thing being possible due the Auto-Segmentation Criterion (ASG) training criteria, which is an improvement over CTC, both being largely explained in Section 2.4.2. Details of the whole architecture are shown in the left side of Fig. 5.

3.6 Facebook TDS-S2S

This model [69] is another Facebook end-to-end approach for the LibriSpeech task, implemented in Wav2Letter framework. Similar to the Facebook CNN-ASG system, the system is composed by a single neural network, a lexicon and it supports a plugged-in language model of the same types, convolutional [56] or n-gram, but in our analysis we considered the second one.

The input of the system is identical with that from the Facebook CNN-ASG approach: 80-dimensional filterbank vectors extracted on-the-fly from audio frames of 25 ms length and 10 ms shift excepting the size of the filterbanks, which is equal to 80 in this case. The sequence length is 240 frames, equivalent to 2.4 seconds processed at once.

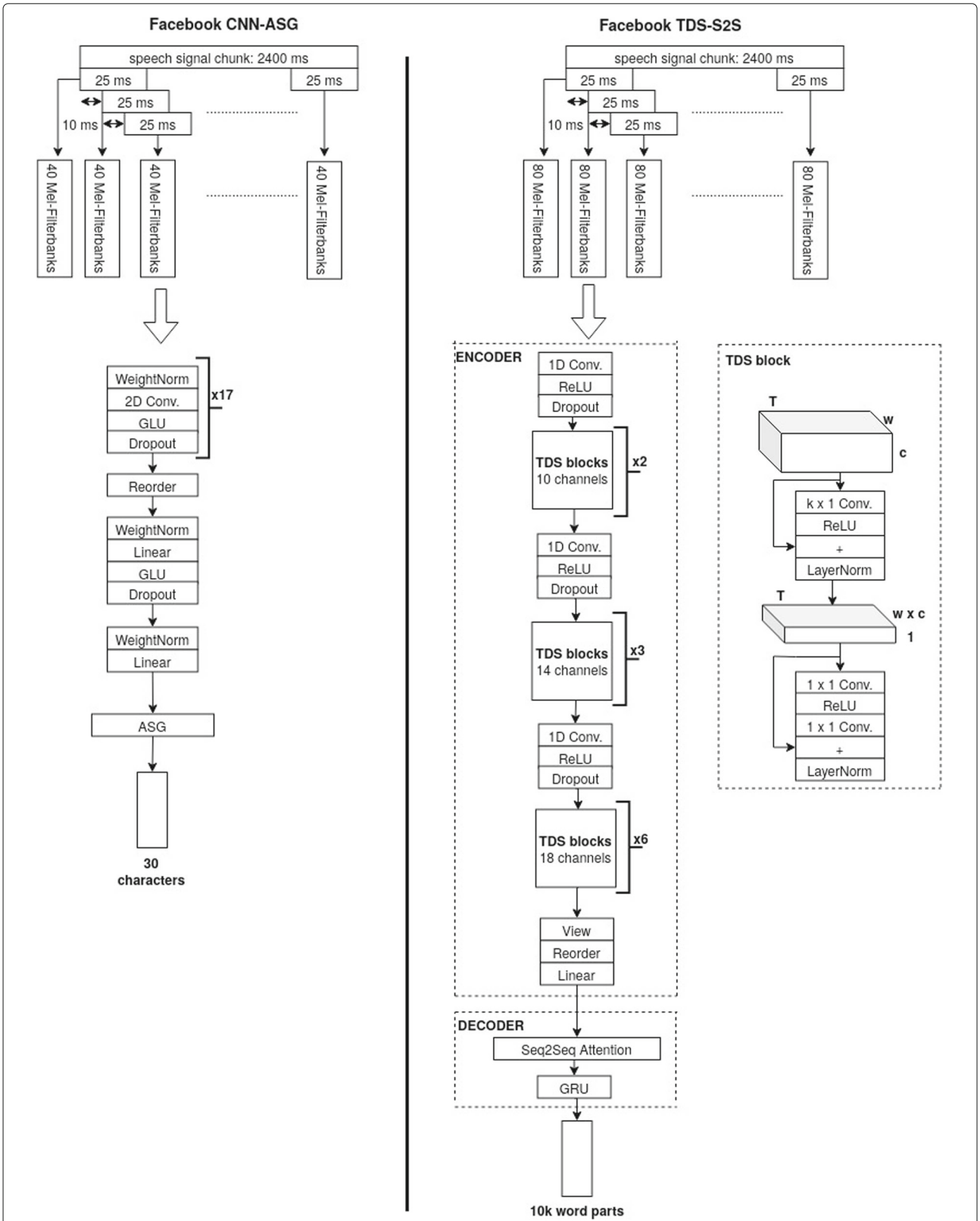

Fig. 5 Facebook Wav2Letter networks: the fully convolutional architecture with ASG loss function (left) and the encoder-decoder with time-depth separable (TDS) blocks (right)

The network is a sequence-to-sequence attention based encoder-decoder. The encoder is represented by a time-depth separable (TDS) convolutional neural network, described in Section 2.4.1. The decoder is a simple recurrent layer, based on Gated Recurrent Units (GRU) [49]. This is a recurrent cell, similar to the LSTM, but with only 3 gates, missing the output gate. The convolutional encoder has an advantage over the recurrent approach, because of the ease with it can be parallelized.

The encoder architecture comprises 11 TDS blocks and 3 interleaved, sub-sampling, 2D convolutional layers: one before the 1st TDS block and the others before the 3rd, respectively the 6th TDS block. The input and the output of each 2D convolution have the shape T x w x c, where T is the time length, w is the feature length and c is the number of channels.

In the time domain, these convolutional layers are performing a sub-sampling operation, halving the sequence length after each of them, due the stride equal to 2. The total sub-sampling factor is 8. The filter size is always 21, while the padding has the value equal to 10. In the feature domain, the features size remains all the time equal to 80, because the convolution uses a filter and a stride both equal to 1, while the padding is 0. At the same time, each sub-sampling brings an output channel increase, due to the time compression. They have values equal to 10, 14 and 18.

Each TDS block is composed by a 2D convolution, over time and features space, similar to the one previously described, but without performing a sub-sampling in time domain. It is followed by a ReLU layer as well as a layer applying a normalization technique. The TDS block contains also 2 convolutional layers with 1x1 kernel, acting like a fully connected layer. These layers are separated by a ReLU non-linearity in between. After the last one is applied a layer normalization. They take an input of shape T x 1 x wc, resulting a same size output. All the time the number of input and output channels, the filters length and the stride values are equal to 1, while the paddings are 0.

After the last TDS block, a reorder layer interchanges the time and features dimensions. This is followed by a linear layer, which takes an 1440-dimensional input and provides an 1024-dimensional output. The output of the encoder is represented by word embeddings.

The GRU decoder has the hidden size equal to 512 and it takes the 1024-dimensional output of the encoder. It has integrated the attention mechanism which performs the alignment. The objective function is a simple log probability over the words sequence. Finally, the network is able to classify over almost 10k word parts, representing the output token set. As in the previous cases, the use of a language model will constrain the output to its vocabulary size, respectively 200k words. The architecture and its component blocks are illustrated on the right side of Fig. 5.

3.7 Nvidia Jasper

Jasper [70] is an end-to-end implementation in OpenSeq2Seq toolkit from Nvidia, created as an approach for the LibriSpeech task. The system comprises a single neural network, without the need of a phonetic model. It takes preprocessed features as input and provides character at the output. The framework provides the possibility of integration with a probabilistic or a Transformer-XL neural network language model [57], as mentioned in Section 2.5.2.

The input of the network is represented by 64-dimensional log-filterbanks, extracted using a 20 ms frame length with 10 ms shift, while the sequence length processed at a time is 160 frames, equivalent to 1.6 seconds.

This network is based on a fully 1D convolutional architecture, which uses deep residual connections. They works as a bypass over the convolutional blocks, avoiding the vanishing gradient problem. The convolutions are only in the time-domain, being similar to a time-delay network.

The architecture is a 10x5 Jasper network, composed of 10 blocks, each one having 5 sub-blocks. Each sub-block performs 1D convolution, batch normalization, ReLU and dropout. The convolution is applied on time domain, using a filter having the same value for two consecutive blocks, but whose size increases, having in turn values of 11, 13, 17, 21, 25. The stride always has a value of 1, and the padding is set so that the length of the output sequence matches the length of the input sequence. All sub-blocks in a block have the same number of the output channels, this number being the same for two consecutive blocks, but it grows with the depth of the network, having values of 256, 384, 512, 640 and 768. The residual connections are represented by 1x1 convolutions followed by batch normalization. They link the input of each sub-block to the output of the block. Therefore, there are 5 residual connections in each block.

The network starts with a pure convolutional layer and it ends with two others. The first one learns 256 channels from 1 input channel, while the last two learn 896 channels from 768 and 1024 from 896. The last layer, a fully-connected layer, performs a 1x1 convolution, where the number of the output channels is 28. This number corresponds to the characters set, over which is provided a probability distribution. Therefore, the network is trained using the CTC criteria, making possible a character based output of the network. The final output of the system is given by the language model, the same 4-gram with 200k words was used in our experiments.

A new optimizer, called NovoGrad [84], is used in this work. This is similar to Adam, but it computes the second

moments per layer, instead of per weight. It helps the network to be more stable and the memory consumption is halved, compared to Adam. The entire system is depicted in the let side of Fig. 6.

3.8 Nvidia QuartzNet

QuartzNet [85] is an end-to-end implementation in the NeMo toolkit from Nvidia, designed as a more efficient variant of Jasper in terms number of parameters and operations. As with Jasper, the system comprises a single neural network that takes preprocessed features as input and provides character as output, and integration with either a probabilistic or a neural network language model is supported.

The input of the network is represented by 64-dimensional log-filterbanks, extracted using a 20 ms

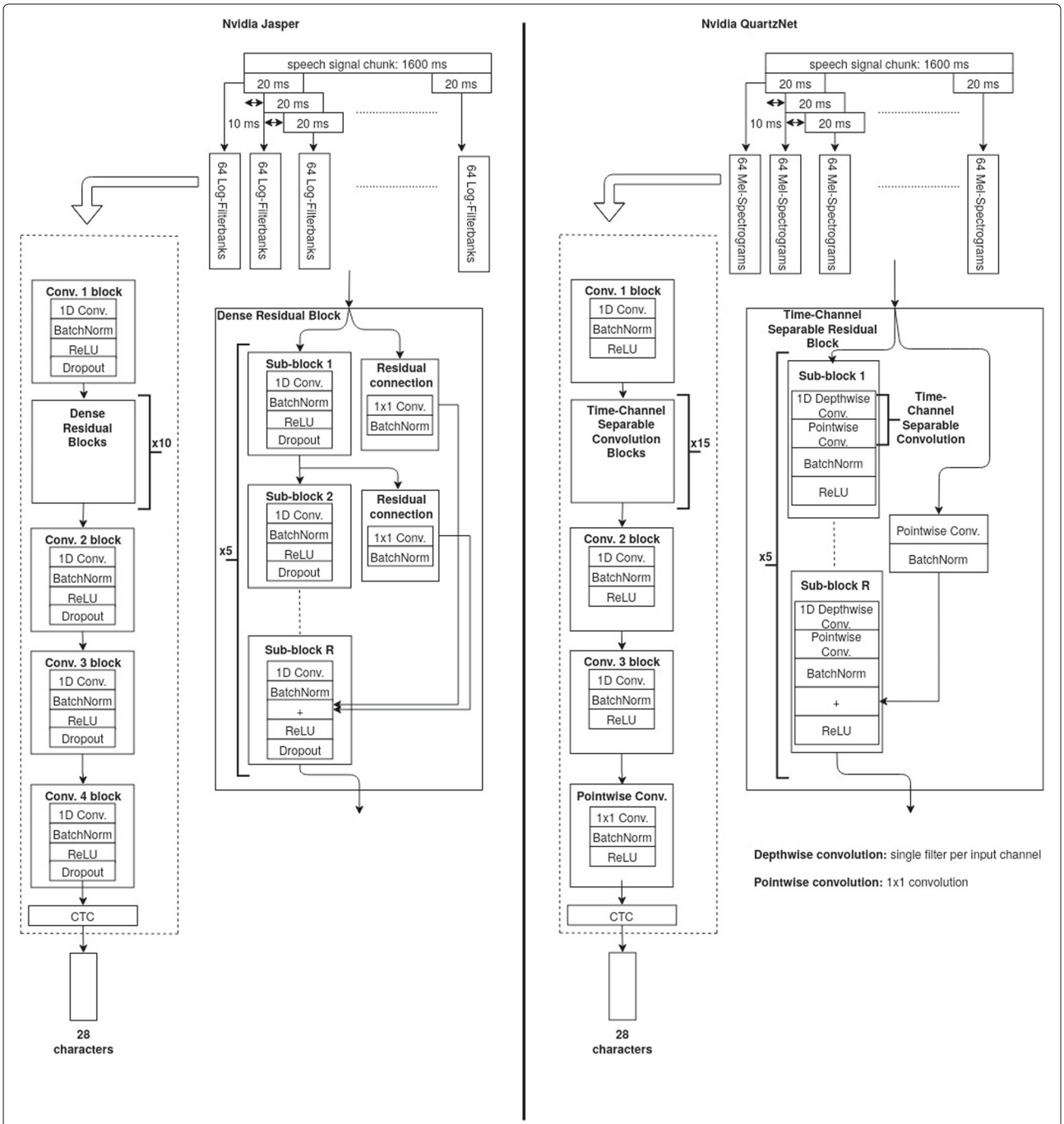

Fig. 6 Nvidia time convolutional networks: Jasper with dense residual blocks (left) and QuartzNet with time-channel separable residual blocks (right)

frame length with 10 ms shift, while the sequence length processed at a time is 160 frames, equivalent to 1.6 seconds.

This network is based on a fully 1D convolutional architecture with residual connections. The main difference with Jasper is the introduction of time-channel separable convolutions, a variation of time-depth separable convolution described earlier in Section 2.4.1.

The architecture we consider is the largest and most accurate variant presented in [85], a 15x5 QuartzNet network.

4 ASR comparison and evaluation. Case study on LibriSpeech

In this section, the implementations presented above, which are specific for the LibriSpeech ASR task, are now evaluated and compared in terms of accuracy and hardware requirements.

These were first analyzed at the system level, and then especially at the neural network component level. Both Kaldi based implementations are multi-component systems, while the other implementations are end-to-end systems. In the first case, the acoustic model, the phonetic dictionary the language model are different components that work together as a system. The acoustic model is a hybrid: time-delay neural network (TDNN) + Hidden Markov Model (HMM). In the second case, a single neural network performs the phonetic and the linguistic modeling, in addition to the acoustic modeling. While in the multi-component systems, the decoding language model is mandatory and only the rescoring language model being optional, in the end-to-end systems also the decoding language model is optional. All the implementations analyzed and tested by us use different hand-crafted features as input. The Kaldi based implementations are the only ones using a combination of two kind of features, MFCCs and i-vectors, while the others are using a single type of features. From the point of view of the network architectures, different variations of the convolution and recurrent networks are used. Multi-component systems use cross-entropy or chain loss objective functions, based on the LF-MMI cost function, while end-to-end systems use more complex mechanisms: sequence to sequence attention, CTC or ASG. They work as loss functions, but also as a HMM, performing the task of aligning the sequences. In terms of output, the neural networks in hybrid approaches are providing posterior probabilities of phonetic units, while the other networks output characters or word parts. At the system level, all systems were used in combination with a probabilistic, 4-gram language model, with a vocabulary of 200k words, this value representing the final output size of the system. A summary of the characteristics of each architecture can be found in the Table 4.

4.1 Evaluation of model complexity

The purpose of the complexity assessment is to know which of the studied architectures are suitable for embedded systems. In an embedded system that is constrained by computational power and memory, an ASR that has few operations, activations and parameters can be integrated. We further describe how we performed the complexity computation. Therefore, the worst case scenario is considered, when all the parameters of the network were kept in memory throughout the inference.

To determine the complexity of each algorithm (model size, number of operations and activations), several operations were performed: (i) the source code was analyzed, (ii) the log files were inspected at the time of inference and (iii) the inference was run step by step in debugger mode. We summarized the information about each layer, such as its type, the input and output dimension, as well as other layer specific additional details. Based on these, we calculated the complexity corresponding to each layer. Table 5 presents the formulas used to perform these calculations.

The number of parameters represents the number of weights learned by the network. In fully connected layers, this is obtained as the product of the input size and the output size of the layer. Time-delay layers are computed similarly, whereby this product being further multiplied by the context size, representing the number of how many vectors at different time frames are considered. In convolutional layers, the number of parameters is given by the multiplication between the filter size (which can be unidimensional or bidimensional) and the number of input and output filters. In the recurrent layers, we used a multiplication formula of four factors. The first factor is the sum between the input size (the features size or the output of the previous size) and the size of the actual recurrent layer. The second factor is the output size of the layer, usually being equal to the size of the recurrent layer. The number of gates depends on the recurrent cell type: 1 for RNN, 3 for GRU, 4 for LSTM. Finally, the fourth factor has the value equal to 1 or 2, indicating whether the layer is unidirectional or bidirectional.

The multiply-accumulate operation (MAC) is defined as the product of two numbers, which is added to an accumulator. In our context, this represents the matrix multiplications that take place in neural networks. The formula is similar to the one used in the calculation of the parameters, in addition being multiplied by the feature vector size and the temporal length of the sequence processed by the network.

The total number of operations (Ops) is equal to twice the number of MACs, because each of them involves a multiplication operation and a summation operation.

The number of activations represent the number of outputs of each layer. In fully connected and time-delay layers, this is obtained as the output size multiplied by

Table 4 Comparison of ASR systems

ASR system	Kaldi TDNN	Kaldi CNN-TDNN	PaddlePaddle Deep-Speech2	Facebook CNN-ASG	Facebook TDS-S2S	RWTH Returnn	Nvidia Jasper	Nvidia QuartzNet
System type	HMM-based	HMM-based	E2E NN	E2E NN	E2E NN	E2E NN	E2E NN	E2E NN
Multi-component vs. single NN	AM (NN) + PD + LM + *[Op. LM rescr.]*	AM (NN) + PD + LM + *[Op. LM rescr.]*	Single NN + *[Op. LM]*	Single NN + *[Op. LM]*	Single NN + *[Op. LM]*	Single NN + BPE encoding list + *[Op. LM]*	Single NN + *[Op. LM]*	Single NN + *[Op. LM]*
Speech features	3 frames x 40 MFCCs + 100 i-vectors	1 frame x 40 fbanks + 200 i-vectors	160 frames x 160 log-spectrograms	240 frames x 40 Mel-fbanks	240 frames x 80 Mel-fbanks	200 frames x 40 MFCC	160 frames x 64 Log-fbanks	160 frames x 64 Mel-spectrograms
NN architecture	Time delay	Conv. + Time delay	Conv. + Bi-RNN	Conv. + GLU	TDS-GRU Enc. - Dec.	LSTM Enc. - Dec. +Attention	Time delay	Time-channel separable conv.
Layers	1 x TDNN 16 x TDNN-F	6 x CNN 12 x TDNN-F	2 x CNN 2 x Bi-RNN	17 x CNN 1 x GLU	1 x CNN + 2 x TDS 1 x CNN + 3 x TDS 1 x CNN + 6 x TDS 1 x GRU	6 x LSTM 1 x Attention	1 x CNN 10 x Dense residual 3 * CNN	1 x CNN 15 x TCS Conv. 2 x CNN 1 x 1 conv.
Output	AM: 6k posteriors + LM: 200k words		NN: 30 chars NN: * words + LM: 200k words		NN: 10k word parts NN: * words + LM: 200k words		NN: 28 characters NN: * words + LM: 200k words	
Loss function	LF-MMI + CE		CTC	ASG	S2S Attention	S2S Attention + CTC	CTC	
Model size[$*10^6$]	20	18	49	208	38	187	333	18.8
Operations per frame [$*10^6$]	41	63	105	22k	15	125	42k	1.8k
Activations per frame [$*10^3$]	44	51	13	1k	9	38k	3k	3.5k

The systems are compared in terms of (i) type: hybrid, HMM-based, versus end-to-end neural network (E2E NN), (ii) component types: multi-component versus single neural network with additional, optional language model, (iii) speech features, (iv) neural network architecture, including the loss function, (v) output size and type, and (vi) model complexity, expressed in terms of model size, number of activations and number of operations required to process one frame of speech. Each system is described in terms of architecture, complexity and hardware requirements

Abbreviations in the table are the following: *AM* Acoustic Model, *ASG* Auto Segmentation Criterion, *ASR* Automatic Speech Recognition, *BPE* Byte-pair encoding, *CE* Cross-entropy, *Conv.* Convolutional, *CNN* Convolutional Neural Network, *CTC* Connectionist Temporal Classification, *Dec.* Decoder, *E2E* End-to-End, *Enc.* Encoder, *GLU* Gated Linear Unit, *GRU* Gated Recurrent Unit, *HMM* Hidden Markov Model, *LF-MMI* Lattice-Free Maximum Mutual Information, *LM* Language Model, *LSTM* Long-Short Term Memory, *MFCC* Mel-Frequency Cepstral Coefficients, *NN* Neural Network, *OOV* Out-of-vocabulary, *Op.* Optional, *PD* Phonetic Model, *rescr.* rescoring, *RNN* Recurrent Neural Network, *RWTH* Rheinisch-Westfälische Technische Hochschule Aachen (Aachen University), *S2S* Sequence-to-Sequence, *TDNN* Time-Delay Neural Network, *TDNN-F* Factored Time-Delay Neural Network, *TDS* Time-Depth Separable

Table 5 Formulas used to calculate the complexity of the networks

Network complexity	Formula
Parameters (model size) in fully connected layers	*input size* $*$ *output size* $+$ *bias*
Parameters (model size) in time-delay layers	(*input size* $*$ *output size* $+$ *bias*) $*$ *context size*
Parameters (model size) in convolutional layers	(*filter size* $*$ *number of input filters* $+$ *bias*) $*$ *number of output filters*
Parameters (model size) in recurrent layers	(*input size* $+$ *recurrent layer size*) $*$ *output size* $*$ *number of gates* $*$ *directionality factor*
Multiply-accumulate operations (MACs)	*parameters* $*$ *features vector length* $*$ *sequence length in time*
Operations (Ops)	*MACs* $*$ 2
Activations in fully connected layers and time-delay layers	*output size* $*$ *output sequence length*
Activations in convolutional layers	*output size* $*$ *number of output filters* $*$ *output sequence length*
Activations in recurrent layers	*recurrent layer size* $*$ *output sequence length* $*$ *directionality factor*

the output sequence length in time. In the case of convolutional layers, the formula is similar, with the difference that the out size could be unidimensional of bidimensional, multiplied also by the number of output filters. The activations in the recurrent layers are obtained by multiplying the size of the recurrent layer, equal to the output size, the temporal sequence length and the directionality factor, being equal to 1 or 2.

4.2 Comparison of ASR systems in terms of model complexity

The size of the various models described in the previous section varies between tens of millions (18 M for Kaldi CNN-TNN and Nvidia QuartzNet) and hundreds of millions (333 M for Nvidia Jasper) of parameters.

Kaldi TDNNs involve learning a smaller number of weights. This is thanks to the medium number of layers (17 and 18), as well as the constant dimensions of the inputs and outputs of the layers throughout the entire network. The time-channel separable convolutions in Nvidia QuartzNet are also leading to a small number of parameters.

Although about double than Nvidia QuartzNet and Kaldi-based systems, PaddlePaddle's DeepSpeech2 implementation and the TDS-S2S from Facebook are economical in terms of learnable parameters. For DeepSpeech2, the reduced number of layers in the encoder (2 convolutional and 3 recurrent), as well as the size (2048) and type of the recurrent ones (without gating mechanisms), directly influence the number of parameters. In the case of the second one, the number of filters varies between 10 and 18 during all the 11 TDS blocks from the encoder, while the size of the filters remains constant. The convolutions are interleaved with fully connected layers.

The recurrent encoder-decoder with attention in RWTH RETURNN is among the models with a large number of parameters (187 M). This is given by the 6 recurrent bidirectional layers in the encoder, each having the size of 1024. More than that, the recurrent cells are LSTMs and each of the 4 characteristic gates implies additional parameters. Also, a contribution in this regard is provided by all the dependencies between the components of the attention mechanism, as well as the recurrent structure of the decoder.

The Facebook CNN-ASG implementation has a large number of parameters. Essential is the large number of filters, between 400 and 1816, which steadily grows along the 17 time-convolutional blocks. The dimensions of the filters have the same increasing character from layer to layer, the smallest having length of 13, and the largest, 29.

The most expensive implementation in terms of model size is, by far, Nvidia Jasper. Although this is also a time-delay network, the large number of parameters is given by the depth of the network. There are 10 dense residual blocks and 4 simple convolutional layers, resulting a total of 54 layers and 50 residual connections. Each block in turn consist of 5 repetitive convolutional sub-blocks. Also, the number of filters used is very high, gradually increasing from 256 to 1024.

From the point of view of the number of operations, these were scaled at frame level and they are up to the giga-order. This is somewhat correlated with the number of parameters: a large number of parameters leads to a large number of operations and vice versa. The computation of the number of operations is crucially influenced by the temporal length of the sequence, taking into account that the network does not process all the data at once, but sequences with a certain length. Because the networks process input sequences of different durations, the number of operations has been scaled, so that the data corresponds to the processing of a single frame, and the results are comparable. The fewest operations are required in the Facebook TDS-S2S implementation: the sequence length decreases with advancement towards the upper layers of the network. The same thing happens in other convolutional implementations, such as in the convolutional layers of DeepSpeech2. The recurrent layers usually retain the same temporal length of the sequence. An exception

occurs in the recurrent layers from RWTH RETURNN implementation. A pooling operation with a factor of 2 is applied, halving the size of the temporal sequence after each one of the 3 recurrent layers at the beginning of the network. QuartzNet performs operations that are one order of magnitude larger than in the case of the previous architectures. The number of operations is quite high in the Facebook CNN-ASG fully convolutional algorithm. Although the temporal length of the sequence decreases to the upper layers of the network, this is slightly higher in comparison with the other algorithms.

By far most operations are involved in Jasper implementation from Nvidia. The difference is up to 3 orders of magnitude compared to the other networks. The time sequence is kept constant, with the help of padding. Also, the depth of the network plays an important role.

From the point of view of the number of activations, they represent the total dimension of the layers' outputs of each network. These are of the order of millions, the least activations are found in the Facebook TDS-S2S implementation, followed closely by PaddlePaddle DeepSpeech2 implementation, due to the small size of the network, while the most activations are found in the implementations from Nvidia.

To summarize this analysis, we can certainly conclude that Nvidia Jasper is the most complex model: largest number of parameters and largest number of operations needed to process a frame of speech. Following closely are the recurrent encoder-decoder from RWTH and the fully convolutional CNN-ASG model from Facebook. With similar model sizes (187M vs. 208M), the first requires significantly more memory to store the 38M activations per frame, while the latter requires significantly more processing power to perform the 22M operations per frame. QuartzNet has a small number of parameters, comparable to Kaldi-based architectures, but in terms of memory it has a medium load, due to the high number of activations. The number of operations is also high, but lower than in the case of Nvidia Jasper or Facebook CNN-ASG architectures.

On the other end, Kaldi's implementations are the lightest models, with only around 20M parameters to be stored in memory, and fastest models, with around 40M - 60M operations to be performed per each speech frame. Facebook TDS-S2S model is also following closely, with a slightly larger model size to put pressure on the system's memory, while requiring three to four times less processing power to process one speech frame.

4.3 Comparison of ASR systems in terms of performance

The previous sections presented in detail the ASR models. We described the work flow and the various components of the systems, emphasizing the complexities of the systems in terms of memory and processing power requirements. The current section aims to analyze the performance of the ASR systems.

The performance evaluation is conducted in terms of word error rate (WER [%]) at system level for each implementation. This is calculated as the total number of transcription errors (insertions, substitutions and deletions), relative to the total number of words in the groundtruth. The error rates presented in Table 6 are either reported by the authors in scientific papers or provided in technical reports on the frameworks repositories.

All the models presented in Table 6 were trained on the training subsets in LibriSpeech, comprising a total of 960 hours of speech (see Table 7). Some ASR systems are able to produce high-quality transcriptions without the need for an additional language model. Therefore, we present results in two scenarios:

Table 6 Comparison of ASR systems in term of performance

ASR system	WER[%]			
	without LM		n-gram LM	
	test-clean	test-other	test-clean	test-other
Kaldi TDNN [64]	-	-	3.85	9.57
Kaldi CNN-TDNN [65]	-	-	3.87	9.42
PaddlePaddle DeepSpeech2 [66]	10.70	30.00	6.03	20.29
RWTH Returnn [67]	4.71	15.17	4.67	15.16
Facebook CNN-ASG [68]	-	-	4.82	14.54
Facebook TDS-S2S [69]	5.36	15.64	4.21	11.87
Nvidia Jasper [70]	3.86	11.93	3.19	9.03
Nvidia QuartzNet [71]	3.90	11.28	2.98	8.38

Performance is expressed in terms of the word error rate (lower is better). The evaluation is performed on two LibriSpeech subsets: test-clean and test-other. For the frameworks which allow this, the evaluation is performed in two scenarios: with or without an external language model

Table 7 Librispeech corpus: the various training and evaluation subsets and their size

Purpose	Set	Size [h]
Training	train-clean-100	100
	train-clean-360	360
	train-other-500	500
Evaluation	test-clean	5.4
	test-other	5.1

- the end-to-end neural network used solely, without an additional language model for rescoring (left side of Table 6) and
- the full ASR system used in conjunction with an external language model: a probabilistic non-pruned 4-gram (*fglarge*) [86] (right side of Table 6).

Note that Facebook implementations also work in the absence of a language model, but the results for this scenario were not reported so far.

The evaluation is performed on the two LibriSpeech test datasets: test-clean, which comprises clean speech, and test-other, which comprises speech recorded in more challenging acoustic environments (see Table 7).

The first conclusion that can be drawn is that the complex models (RWTH RETURNN and Nvidia Jasper) obtain similar results regardless of whether an external language model is used or not. By contrast, Facebook TDS-S2S and PaddlePaddle DeepSpeech2, which also have the ability to work without an external LM, perform poorly in this situation.

The best results are obtained with Nvidia Quartznet, 2,98% WER on clean speech and 8.38% on non-clean speech. They are closely followed by the Nvidia Jasper and Kaldi based systems. The systems from Facebook and RWTH RETURNN follow in this hierarchy, while PaddlePaddle DeepSpeech2 is at the end of this ranking.

4.4 Trade-offs between ASR performance and hardware requirements

This subsection concludes the evaluation section by discussing the various trade-offs between ASR system performance and hardware requirements for the various systems analyzed. While in previous sections we compared the ASR systems with respect to model complexity and performance separately, we are now interested to see how the model complexities translate to hardware requirements. Moreover, it is important to understand if stronger hardware constraints lead to improved ASR accuracy or not. Finally, this analysis should pinpoint the ASR systems that meet the requirements for embedded speech applications.

Based on the data in Table 4, we estimated the memory requirements for each ASR system, by making the following assumptions: (i) the model parameters and activations that need to be stored in the memory are 4B numbers and (ii) all ASR system need to store in the memory the models and at least the network activations required to process 1 second of speech. The memory load results are presented in Table 8.

In Table 8 we also present the required throughput of the hardware system. The throughput is the number of operations per second that the hardware should be able to perform in order to process speech data in real time (i.e. process one second of speech in one second). Finally, for the sake of simplicity, we only express the ASR performance in terms of the WER obtained on the most popular LibriSpeech evaluation scenario: the ASR uses an external LM for language rescoring and the evaluation is performed on text-clean (i.e. the subset that contains speech recorded in clean acoustic environments).

The data shows that RWTH RETURNN, Nvidia Jasper, Nvidia QuartzNet and Facebook CNN-ASG are prohibitive with respect to memory load. They require between 14x and 235x more memory than the lightest system (Kaldi CNN-TDNN). On the opposite side, Kaldi-based systems, PaddlePaddle DeepSpeech2 and Facebook TDS-S2S require similar amounts of memory: between 100 and 200 MB[1].

The trade-off between ASR performance and memory requirements is presented as a trade-off diagram in Fig. 7 (left). Although Nvidia QuartzNet requires a large amount of memory, it rests on the Pareto front thanks to its low WER (high performance). Kaldi-based systems are both on the Pareto front because they dominate each other (i.e. CNN-TDNN is better in terms performance, and TDNN is better in terms of memory requirements).

With regard to computational power requirements, the data clearly shows that Nvidia Jasper and Facebook CNN-ASG are, again, prohibitive. They make between 1400x and 2800x more operations than the fastest system (Facebook TDS-S2S). Facebook TDS-S2S is also significantly faster (2.7x) than its successor (i.e. Kaldi TDNN) and the subsequent systems.

The trade-off between ASR performance and memory requirements is presented as a trade-off diagram in Fig. 7

[1] This is only the amount of memory required to load the neural model and store all the activations of the network for processing 1 second of speech. More memory might be needed for other components, such as the language model etc.

Table 8 ASR performance vs. hardware requirements trade-off

ASR system	Performance	Hardware requirements		System on Pareto frontier
	WER[%]	Memory [MB]	Throughput [GOPS]	
Kaldi TDNN	3.85	106	4.1	✓
Kaldi CNN-TDNN	3.87	103	6.3	✓
Paddle Paddle DeepSpeech2	6.03	204	10.5	✗
RWTH Returnn	4.67	23548	12.5	✗
Facebook CNN-ASG	4.82	1432	2200	✗
Facebook TDS-S2S	4.21	157	1.5	✓
Nvidia Jasper	3.19	3132	4200	✗
Nvidia QuartzNet	2.98	2169	180	✓

Performance is expressed in terms of the word error rate obtained on LibriSpeech test-clean dataset (lower is better). Hardware requirements are expressed in terms of memory load, in Mega bytes (MB), and minimum throughput, in Giga operations per second (GOPS). Note that for memory load we only took into account the amount needed to load the neural model and store all the activations of the network for processing 1 second of speech. More memory might be needed for other components, such as the language model etc. For throughput we only considered the operations required to pass the speech through the network. More operations might be needed for other processes, such as language rescoring etc

(right). Nvidia QuartzNet and Facebook TDS-S2S are extremes on the Pareto front: the first has the best performance, while the latter requires the least amount of operations to process on second of speech. As a trade-off between the two, Kaldi TDNN is also on the Pareto front: it is significantly faster than Nvidia QuartzNet and more accurate than Facebook TDS-S2S.

Finally, we also analyzed the trade-off in a 3-dimensional scenario: for an embedded application one would be interested in a simultaneous trade-off between ASR performance, memory and computational power requirements. The results are presented in Table 8 (see last column). Among the eight systems that were analyzed, Nvidia QuartzNet, Facebook TDS-S2S and Kaldi CNN-TDNN are extremes on the Pareto front, being the best systems in terms of performance, throughput requirements and respectively memory requirements. As a trade-off between the three, Kaldi TDNN still provides Pareto optimal design points, dominating each of the other three systems in two measures.

5 Conclusion

This article presented an overview of the fundamentals of automatic speech recognition systems and their evolution over the last years. The general architecture of an ASR system was presented, as well as the various approaches for each component part. We summarized the most popular types of speech features and the way that they can be extracted and improved. We passed over the acoustic modeling algorithms, from traditional probabilistic models, which were replaced by neural networks in hybrid approaches, to end-to-end models using pure neural networks without an alignment model. We also presented the main approaches of language modeling for ASR. Based

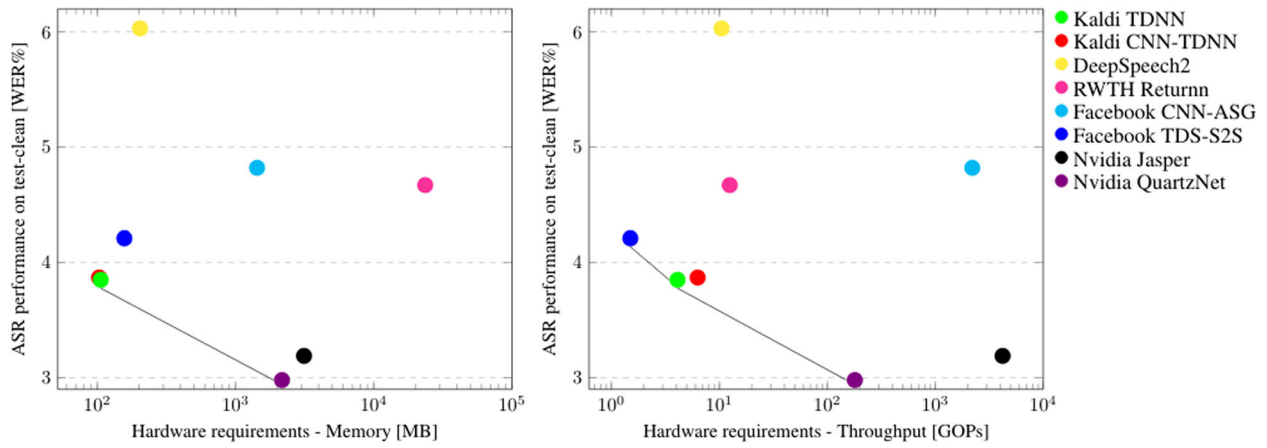

Fig. 7 Trade-off between ASR performance and memory requirements (left) and between ASR performance and throughput requirements (right). Memory load is expressed in Mega bytes (MB). Throughput is expressed in Gigaoperations per second (GOPS)

on this overview, several conclusions can be drawn with respect to the current trends in the field of automatic speech recognition.

Although pipeline ASR systems represented the state of the art up to now, end-to-end systems are on an ascending trend and will most likely replace them successfully in the near future. Our analysis showed that only one end-to-end system (Nvidia QuartzNet) has managed to surpass the state of the art pipeline system (Kaldi TDNN) in terms of accuracy, having a similar number of parameters, but with a greater cost in terms of number of operations required to process one frame of speech.

From the point of view of the acoustic features, many implementations, including state of the art systems, still use traditional features, such as MFCCs or i-vectors.

Kaldi, for example, combines MFCC and i-vector features in various ways, depending on the neural layers that process the features further. Moreover, in Kaldi several feature transforms and data augmentation techniques (e.g. speed and noise perturbations) are used. However, we can clearly see the attempt to migrate from hand-crafted features towards the raw waveform. Many networks use special input layers that act like a feature extractor front-end. Although the trend is clear and many attempts were made to perform automatic speech recognition from the raw waveform, current end-to-end ASR implementations still use spectrograms or filterbanks as input.

Acoustic modeling is performed in all state of the art ASR implementations using neural networks. Most approaches use recurrent networks and convolutional networks. The disadvantage of the recurrent networks consists in the difficulty of parallelization. Moreover, the bidirectional networks require special tricks if they are to be used for online transcription, as the entire signal is not available in this scenario. The use of residual connections became a common practice, thus improving the propagation of data over the network, sometimes working as a shortcut over some layers.

While the acoustic units modelled in the state of the art systems are phones or subphonetic units (e.g. senones), the trend is to migrate away from these intermediate representations and to model directly characters, word parts or even words. As there are already effective implementations that output characters or word parts (6 of the 8 implementations analyzed) it is clear that the phonetic dictionary in a traditional pipeline ASR system will soon become obsolete.

In terms of language modeling, pipeline systems, as those based on Kaldi, are compulsory depending on a language model. End-to-end systems can optionally use a language model as an add-on. Its use leads to better results, but it is not crucial. All results can be further improved by using a rescoring model, which takes over and corrects the initial transcript. Regarding the type of the language model, probabilistic models, n-grams, are still used and they are very popular, but the neural approaches, with convolutional or recurrent language models, are superior in terms of accuracy, although more expensive in terms of computation.

Apart from the overview on automatic speech recognition, we conducted an in-depth analysis of eight different ASR implementations for LibriSpeech ASR task in order to identify the frameworks which might be suitable for integration in embedded applications with a minimal drop in performance. We evaluated the TDNN and CNN-TDNN architectures from Kaldi, RETURNN attention-based encoder-decoder from RWTH, fully convolutional CNN-ASG and TDS-S2S from Facebook, PaddlePaddle DeepSpeech2 from PaddlePaddle, Jasper dense-residual and QuartzNet from Nvidia. We described and compared the various features used as input, the various types and sizes of layers and blocks of layers, the loss functions, the various output types and their link to output words etc. The analysis aimed at offering an insight into whether these models are suitable or not for integration in embedded applications. To this end, we first expressed the complexities of the models in terms of model size, number of activations and number of operations required to process one frame of speech. Finally, we translated these metrics into hardware requirements, such as memory load and minimum throughput, metrics that can be directly to decide which ASR implementation suits certain hardware constrains. To the best of our knowledge, this is the first article that presented such an analysis.

The conclusions that aroused from this analysis are very interesting. We showed that some end-to-end implementations (i.e. RWTH RETURNN, Nvidia Jasper, Nvidia QuartzNet and Facebook CNN-ASG) are prohibitive for embedded applications due to their memory requirements. They require between 14x and 235x more memory than the lightest system (i.e. Kaldi CNN-TDNN). On the opposite side, Kaldi-based systems, PaddlePaddle DeepSpeech2 and Facebook TDS-S2S require similar amounts of memory.

With regard to computational power requirements, we conclude that Nvidia Jasper and Facebook CNN-ASG are, again, not suitable for embedded applications. They make between 1400x and 2800x more operations than the fastest system (i.e. Facebook TDS-S2S). Facebook TDS-S2S is also significantly faster (2.7x) than its successor (i.e. Kaldi TDNN) and the subsequent systems.

For an embedded application one would be interested in a simultaneous trade-off between ASR performance, memory and computational power requirements. With respect to this, our trade-off analysis showed that Nvidia QuartzNet, Facebook TDS-S2S and Kaldi CNN-TDNN are extremes on the Pareto front, being the best systems in terms of performance, throughput requirements

and respectively memory requirements. As a trade-off between the three, Kaldi TDNN still provides Pareto optimal design points, dominating each of the other systems in two out of the three measures.

Abbreviations

AM: Acoustic Model; ASG: Auto Segmentation Criterion; ASR: Automatic Speech Recognition; BLSTM: Bidirectional Long-Short Term Memory; BPE: Byte-pair encoding; CE: Cross-Entropy; CNN: Convolutional Neural Network; CTC: Connectionist Temporal Classification; DNN: Deep Neural Network; DCT: Discrete Cosine Transform; E2E: End-to-End; EM: Expectation-Maximization; FFT: Fast Fourier Transform; GLU: Gated Linear Unit; GMM: Gaussian Mixture Model; GOPS: Giga Operations per Second; GRU: Gated Recurrent Unit; HMM: Hidden Markov Model; JFA: Joint Factor Analysis; LDA: Linear Discriminant Analysis; LF-MMI: Lattice-Free Maximum Mutual Information; LM: Language Model; LSTM: Long-Short Term Memory; MAC: Multiply-Accumulate Operation; MB: Mega bytes; MFCC: Mel-Frequency Cepstral Coefficients; MFSC: Mel-Frequency Spectral Coefficients; MMI: Maximum Mutual Information; NN: Neural Network; OOV: Out-of-vocabulary; Ops: Operations; PD: Phonetic Model; ReLU: Rectified Linear Unit; RNN: Recurrent Neural Network; RWTH: Rheinisch-Westfälische Technische Hochschule Aachen (Aachen University); S2S: Sequence-to-Sequence; SVD: Single Value Decomposition; TDNN: Time-Delay Neural Network; TDNN-F: Factored Time-Delay Neural Network; TDS: Time-Depth Separable; WER: Word Error Rate; WSJ: Wall Street Journal

Acknowledgements

Not applicable.

Authors' contributions

ALG was responsible for summarizing the methods and writing the manuscript. AP provided guidance for ALG in order to perform the experiments and the subsequent analyzes. ALG and HC organized the manuscript structure and content. MB supevised the entire project and provided suggestions on the writing. All authors have read and approved the final manuscript. The contributions of all authors are considered to be equal.

Authors' information

ALG graduated with a Master's degree from the University Politehnica of Bucharest, Romania, where he currently follows his Ph.D. studies as a member of the Speech and Dialogue research laboratory. The main direction of interest is artificial intelligence applied in speech technology, including speech recognition and speaker recognition. In 2019, he did this work during an internship at Xilinx Research Labs, Dublin.

AP is a Senior Engineer at Xilinx Research in Dublin, Ireland, where he works at the intersection of hardware and software for machine learning acceleration. He earned his Bachelor's degree from Politecnico di Milano, Italy, and his Master's degree from the University of Illinois at Chicago, USA.

HC received the Ph.D. degree in electronics and telecommunications from University Politehnica of Bucharest (2011), where he has served as Associate Professor since 2017. His research interests include machine learning and artificial intelligence, with a special focus on automatic speech and speaker recognition methods. He authored over 75 scientific papers in international conferences and journals. He was awarded the Romanian Academy prize "Mihail Drăgănescu" (2016) for outstanding research contributions in Spoken Language Technology, after developing the first automatic speech recognition system for the Romanian language.

MB is a Distinguished Engineer at Xilinx Research in Dublin, Ireland, where she heads a team of international scientists driving exciting research to define new application domains for Xilinx devices, such as machine learning. She earned her Master's degree from the University of Kaiserslautern in Germany and brings over 25 years of computer architecture, FPGA and board design, in research institutions (ETH Zurich and Bell Labs) and development organizations. She is heavily involved with the international research community serving as the technical co-chair of FPL'2018, workshop organizer (H2RC, ITEM'2020), and member of numerous technical program committees (FPL, ISFPGA, DATE, etc.).

Funding

This work was partly supported by a grant of the Romanian Ministry of Research and Innovation, CCCDI – UEFISCDI, project number PN-III-P1-1.2-PCCDI-2017-0818 / 73PCCDI, within PNCDI III.

Availability of data and materials

Not applicable.

Declarations

Competing interests

The authors declare that they have no competing interests.

Author details

[1]Speech and Dialogue Research Laboratory, University Politehnica of Bucharest, Bucharest, Romania. [2]Research Labs, Xilinx, Dublin, Ireland.

Received: 11 February 2021 Accepted: 23 June 2021
Published online: 21 July 2021

References

1. M. Price, J. Glass, A. P. Chandrakasan, A low-power speech recognizer and voice activity detector using deep neural networks. IEEE J. Solid-State Circ. **53**(1), 66–75 (2017)
2. M. Lee, K. Hwang, J. Park, S. Choi, S. Shin, W. Sung, in *2016 IEEE International Workshop on Signal Processing Systems (SiPS)*. Fpga-based low-power speech recognition with recurrent neural networks (IEEE, Dallas, 2016), pp. 230–235. https://doi.org/10.1109/SiPS.2016.48
3. S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, et al., in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. Ese: Efficient speech recognition engine with sparse lstm on fpga, (2017), pp. 75–84. https://doi.org/10.1145/3020078.3021745
4. B. Liu, H. Qin, Y. Gong, W. Ge, M. Xia, L. Shi, Eera-asr: An energy-efficient reconfigurable architecture for automatic speech recognition with hybrid dnn and approximate computing. IEEE Access. **6**, 52227–52237 (2018)
5. R. Yazdani, A. Segura, J.-M. Arnau, A. Gonzalez, in *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. An ultra low-power hardware accelerator for automatic speech recognition (IEEE, Taipei, 2016), pp. 1–12. https://doi.org/10.1109/MICRO.2016.7783750
6. S. Migacz, in *GPU Technology Conference, vol. 2*. 8-bit inference with tensorrt, (2017), p. 5. https://on-demand.gputechconf.com/gtc/2017/presentation/s7310-8-bit-inference-withtensorrt.pdf
7. Z. Zhang, J. Geiger, J. Pohjalainen, A. E.-D. Mousa, W. Jin, B. Schuller, Deep learning for environmentally robust speech recognition: An overview of recent developments. ACM Trans. Intell. Syst. Technol. (TIST). **9**(5), 1–28 (2018)
8. J. Park, Y. Boo, I. Choi, S. Shin, W. Sung, in *Advances in Neural Information Processing Systems*. Fully neural network based speech recognition on mobile and embedded devices, (2018), pp. 10620–10630. https://dl.acm.org/doi/10.5555/3327546.3327722
9. H. Purwins, B. Li, T. Virtanen, J. Schlüter, S.-Y. Chang, T. Sainath, Deep learning for audio signal processing. IEEE J. Sel. Top. Signal Process. **13**(2), 206–219 (2019)
10. C. Kim, D. Gowda, D. Lee, J. Kim, A. Kumar, S. Kim, A. Garg, C. Han, A review of on-device fully neural end-to-end automatic speech recognition algorithms. arXiv preprint arXiv:2012.07974 (2020)
11. D. Wang, X. Wang, S. Lv, An overview of end-to-end automatic speech recognition. Symmetry. **11**(8), 1018 (2019)
12. C. Shan, J. Zhang, Y. Wang, L. Xie, in *ICASSP*. Attention-based end-to-end speech recognition on voice search (IEEE, 2018), pp. 4764–4768. https://doi.org/10.1109/ICASSP.2018.8462492
13. R. Collobert, C. Puhrsch, G. Synnaeve, Wav2letter: an end-to-end convnet-based speech recognition system. arXiv preprint arXiv:1609.03193 (2016)
14. M. Alam, M. D. Samad, L. Vidyaratne, A. Glandon, K. M. Iftekharuddin, Survey on deep neural networks in speech and vision systems. Neurocomputing. **417**, 302–321 (2020)
15. T. N. Sainath, R. Prabhavalkar, S. Kumar, S. Lee, A. Kannan, D. Rybach, V. Schogol, P. Nguyen, B. Li, Y. Wu, et al., in *ICASSP*. No need for a lexicon? evaluating the value of the pronunciation lexica in end-to-end models (IEEE, 2018), pp. 5859–5863. https://doi.org/10.1109/icassp.2018.8462380
16. C.-C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, E. Gonina, et al., in *ICASSP*. State-of-the-art speech recognition with sequence-to-sequence models (IEEE, 2018), pp. 4774–4778. https://doi.org/10.1109/ICASSP.2018.8462105

17. R. Collobert, A. Hannun, G. Synnaeve, Word-level speech recognition with a dynamic lexicon. arXiv preprint arXiv:1906.04323 (2019)
18. A. Graves, A.-r. Mohamed, G. Hinton, in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. Speech recognition with deep recurrent neural networks (IEEE, 2013), pp. 6645–6649. https://doi.org/10.1109/ICASSP.2013.6638947
19. H. Hadian, H. Sameti, D. Povey, S. Khudanpur, in *Interspeech*. End-to-end speech recognition using lattice-free mmi, (2018), pp. 12–16. https://doi.org/10.21437/Interspeech.2018-1423
20. R. W. Hamming, *Digital Filters*. (Courier Corporation, 1998)
21. A. V. Oppenheim, *Discrete-time Signal Processing*. (Pearson Education India, 1999)
22. S. S. Stevens, J. Volkmann, E. B. Newman, A scale for the measurement of the psychological magnitude pitch. J. Acoust. Soc. Am. **8**(3), 185–190 (1937)
23. S. Davis, P. Mermelstein, Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. IEEE Trans. Acoust. Speech Signal Process. **28**(4), 357–366 (1980)
24. N. Dehak, P. J. Kenny, R. Dehak, P. Dumouchel, P. Ouellet, Front-end factor analysis for speaker verification. IEEE Trans. Audio Speech Lang. Process. **19**(4), 788–798 (2010)
25. P. Kenny, G. Boulianne, P. Ouellet, P. Dumouchel, Joint factor analysis versus eigenchannels in speaker recognition. IEEE Trans. Audio Speech Lang. Process. **15**(4), 1435–1447 (2007)
26. R. D. Lopez-Cozar, M. Araki, *Spoken, Multilingual and Multimodal Dialogue Systems: Development and Assessment*. (Wiley, 2005). https://doi.org/10.1002/0470021578
27. Y. Zhang, M. Alder, R. Togneri, in *ICASSP, vol. 1*. Using gaussian mixture modeling in speech recognition (IEEE, 1994), p. 613. https://doi.org/10.1109/ICASSP.1994.389219
28. S. J. Young, J. J. Odell, P. C. Woodland, in *Proceedings of the Workshop on Human Language Technology*. Tree-based state tying for high accuracy acoustic modelling (Association for Computational Linguistics, 1994), pp. 307–312. https://doi.org/10.3115/1075812.1075885
29. L. E. Baum, T. Petrie, G. Soules, N. Weiss, A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. Ann. Math. Stat. **41**(1), 164–171 (1970)
30. A. Viterbi, Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Trans. Inf. Theory. **13**(2), 260–269 (1967)
31. G. Hinton, et al., Deep Neural Networks for Acoustic Modeling in Speech Recognition. IEEE Signal Proc. Mag. **29**(6), 82–97 (2012)
32. A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, K. J. Lang, Phoneme recognition using time-delay neural networks. IEEE Trans. Acoust. Speech Signal Process. **37**(3), 328–339 (1989)
33. V. Peddinti, D. Povey, S. Khudanpur, in *Interspeech*. A time delay neural network architecture for efficient modeling of long temporal contexts, (2015), pp. 3214–3218. https://academic.microsoft.com/paper/2402146185/reference
34. D. Povey, G. Cheng, Y. Wang, K. Li, H. Xu, M. Yarmohammadi, S. Khudanpur, in *Interspeech*. Semi-orthogonal low-rank matrix factorization for deep neural networks, (2018), pp. 3743–3747. https://doi.org/10.21437/Interspeech.2018-1417
35. O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, D. Yu, Convolutional neural networks for speech recognition. IEEE/ACM Trans. Audio Speech Lang. Process. **22**(10), 1533–1545 (2014)
36. Kaldi Help Google Group: CNN-TDNN vs. TDNN (2020). https://groups.google.com/d/msg/kaldi-help/jsg1Oo4bNGQ/uwvFw5PtBwAJ. Accessed 23 Mar 2020
37. F. L. Kreyssig, C. Zhang, P. C. Woodland, in *ICASSP*. Improved tdnns using deep kernels and frequency dependent grid-rnns (IEEE, 2018), pp. 4864–4868. https://doi.org/10.1109/ICASSP.2018.8462523
38. A. Biswas, E. Yılmaz, F. de Wet, E. van der Westhuizen, T. Niesler, in *Interspeech*. Semi-Supervised Acoustic Model Training for Five-Lingual Code-Switched ASR, (2019), pp. 3745–3749. https://doi.org/10.21437/interspeech.2019-1325
39. C. Zorilă, C. Boeddeker, R. Doddipatla, R. Haeb-Umbach, in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. An investigation into the effectiveness of enhancement in asr training and test for chime-5 dinner party transcription (IEEE, 2019), pp. 47–53. https://doi.org/10.1109/ASRU46091.2019.9003785
40. N. Zeghidour, Q. Xu, V. Liptchinsky, N. Usunier, G. Synnaeve, R. Collobert, Fully convolutional speech recognition. arXiv preprint arXiv:1812.06864 (2018)
41. G. Huang, Z. Liu, L. Van Der Maaten, K. Q. Weinberger, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Densely connected convolutional networks, (2017), pp. 4700–4708. https://doi.org/10.1109/cvpr.2017.243
42. J. Li, V. Lavrukhin, B. Ginsburg, R. Leary, O. Kuchaiev, J. M. Cohen, H. Nguyen, R. T. Gadde, in *Interspeech*. Jasper: An End-to-End Convolutional Neural Acoustic Model, (2019), pp. 71–75. https://doi.org/10.21437/interspeech.2019-1819
43. H. Sak, A. Senior, F. Beaufays, in *Interspeech*. Long short-term memory recurrent neural network architectures for large scale acoustic modeling, (2014), pp. 338–342. https://research.google/pubs/pub43905.pdf
44. F. A. Gers, N. N. Schraudolph, J. Schmidhuber, Learning precise timing with lstm recurrent networks. J. Mach. Learn. Res. **3**(Aug), 115–143 (2002)
45. D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, et al., in *2016 International Conference on Machine Learning*. Deep speech 2: End-to-end speech recognition in english and mandarin, (2016), pp. 173–182. https://academic.microsoft.com/paper/2193413348/reference
46. D. Bahdanau, K. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473 (2014)
47. W. Chan, N. Jaitly, Q. Le, O. Vinyals, in *ICASSP*. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition (IEEE, 2016), pp. 4960–4964. https://doi.org/10.1109/ICASSP.2016.7472621
48. I. Sutskever, et al., Q. Le, Sequence to Sequence Learning with Neural Networks. Adv. Neural Inf. Process. Syst. **27**, 3104–3112 (2014)
49. K. Cho, B. van Merriënboer, D. Bahdanau, Y. Bengio, in *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. On the properties of neural machine translation: Encoder–decoder approaches, (2014), pp. 103–111. https://doi.org/10.3115/v1/w14-4012
50. A. Hannun, A. Lee, Q. Xu, R. Collobert, in *Interspeech*. Sequence-to-Sequence Speech Recognition with Time-Depth Separable Convolutions, (2019), pp. 3785–3789. https://doi.org/10.21437/interspeech.2019-2460
51. A. Graves, S. Fernández, F. Gomez, J. Schmidhuber, in *Proceedings of the 23rd International Conference on Machine Learning*. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks (ACM, 2006), pp. 369–376. https://doi.org/10.1145/1143844.1143891
52. T. Hori, S. Watanabe, Y. Zhang, W. Chan, in *Interspeech*. Advances in joint ctc-attention based end-to-end speech recognition with a deep cnn encoder and rnn-lm, (2017). https://doi.org/10.21437/INTERSPEECH.2017-1296
53. D. Povey, V. Peddinti, D. Galvez, P. Ghahremani, V. Manohar, X. Na, Y. Wang, S. Khudanpur, in *Interspeech*. Purely sequence-trained neural networks for asr based on lattice-free mmi, (2016), pp. 2751–2755. https://doi.org/10.21437/Interspeech.2016-595
54. T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, S. Khudanpur, in *2010 Conference of the International Speech Communication Association*. Recurrent neural network based language model, (2010). https://academic.microsoft.com/paper/179875071/reference
55. M. Sundermeyer, R. Schlüter, H. Ney, in *2012 Conference of the International Speech Communication Association*. Lstm neural networks for language modeling, (2012). https://academic.microsoft.com/paper/2402268235/reference
56. Y. N. Dauphin, A. Fan, M. Auli, D. Grangier, in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. Language modeling with gated convolutional networks (JMLR. org, 2017), pp. 933–941. https://academic.microsoft.com/paper/2963970792/reference
57. Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. Le, R. Salakhutdinov, in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Transformer-XL: Attentive language models beyond a fixed-length context (Association for Computational Linguistics, Florence, 2019), pp. 2978–2988
58. C. Gulcehre, O. Firat, K. Xu, K. Cho, L. Barrault, H.-C. Lin, F. Bougares, H. Schwenk, Y. Bengio, On using monolingual corpora in neural machine translation. arXiv preprint arXiv:1503.03535 (2015)
59. A. Kannan, Y. Wu, P. Nguyen, T. N. Sainath, Z. Chen, R. Prabhavalkar, in *ICASSP*. An analysis of incorporating an external language model into a

sequence-to-sequence model (IEEE, 2018), pp. 1–5828. https://doi.org/10.1109/icassp.2018.8462682
60. A. Sriram, H. Jun, S. Satheesh, A. Coates, in *Interspeech*. Cold fusion: Training seq2seq models together with language models, (2018), pp. 387–391. https://doi.org/10.21437/interspeech.2018-1392
61. S. Toshniwal, A. Kannan, C.-C. Chiu, Y. Wu, T. N. Sainath, K. Livescu, in *2018 IEEE Spoken Language Technology Workshop (SLT)*. A comparison of techniques for language model integration in encoder-decoder speech recognition (IEEE, 2018), pp. 369–375. https://doi.org/10.1109/SLT.2018.8639038
62. T. Mikolov, S. Kombrink, A. Deoras, L. Burget, J. Cernocky, in *Proc. of the 2011 ASRU Workshop*. Rnnlm-recurrent neural network language modeling toolkit, (2011), pp. 196–201. https://academic.microsoft.com/paper/2474824677/reference
63. H. Xu, et al., in *ICASSP*. A pruned rnnlm lattice-rescoring algorithm for automatic speech recognition (IEEE, 2018), pp. 5929–5933. https://doi.org/10.1109/ICASSP.2018.8461974
64. Kaldi TDNN LibriSpeech implementation (2020). https://github.com/kaldi-asr/kaldi/blob/master/egs/librispeech/s5/local/chain/tuning/run_tdnn_1d.sh. Accessed 23 Mar 2020
65. Kaldi CNN-TDNN LibriSpeech implementation (2020). https://github.com/kaldi-asr/kaldi/blob/master/egs/librispeech/s5/local/chain/tuning/run_cnn_tdnn_1a.sh. Accessed 23 Mar 2020
66. PaddlePaddle DeepSpeech2 LibriSpeech implementation (2020). https://github.com/PaddlePaddle/DeepSpeech/blob/develop/model_utils/network.py. Accessed 23 Mar 2020
67. RWTH Returnn LibriSpeech implementation (2020). https://github.com/rwth-i6/returnn-experiments/blob/master/2018-asr-attention/librispeech/full-setup-attention/returnn.config. Accessed 23 Mar 2020
68. Wav2Letter CNN-GLU fully convolutional LibriSpeech implementation (2020). https://github.com/facebookresearch/wav2letter/blob/master/recipes/models/conv_glu/librispeech/network.arch. Accessed 23 Mar 2020
69. Wav2Letter time-domain separable LibriSpeech implementation (2020). https://github.com/facebookresearch/wav2letter/blob/master/recipes/models/seq2seq_tds/librispeech/network.arch. Accessed 23 Mar 2020
70. Nividia OpenSeq2Seq Jasper LibriSpeech implementation (2020). https://github.com/NVIDIA/OpenSeq2Seq/blob/master/example_configs/speech2text/jasper10x5_LibriSpeech_nvgrad.py. Accessed 23 Mar 2020
71. Nvidia QuartzNet implementation (2020). https://github.com/NVIDIA/NeMo/blob/master/examples/asr/configs/quartznet15x5.yaml. Accessed 23 Mar 2020
72. V. Panayotov, G. Chen, D. Povey, S. Khudanpur, in *ICASSP*. Librispeech: an asr corpus based on public domain audio books (IEEE, 2015), pp. 5206–5210. https://doi.org/10.1109/ICASSP.2015.7178964
73. D. B. Paul, J. M. Baker, in *Proceedings of the Workshop on Speech and Natural Language*. The design for the wall street journal-based csr corpus (Association for Computational Linguistics, 1992), pp. 357–362. https://doi.org/10.3115/1075527.1075614
74. A. Rousseau, P. Deléglise, Y. Esteve, in *2014 Language Resources and Evaluation*. Enhancing the ted-lium corpus with selected data for language modeling and more ted talks, (2014), pp. 3935–3939. https://academic.microsoft.com/paper/2251321385/reference
75. J. J. Godfrey, E. C. Holliman, J. McDaniel, in *ICASSP, vol. 1*. Switchboard: Telephone speech corpus for research and development (IEEE, 1992), pp. 517–520. https://doi.org/10.1109/ICASSP.1992.225858
76. C. Cieri, D. Miller, K. Walker, in *2004 Language Resources and Evaluation, vol. 4*. The fisher corpus: a resource for the next generations of speech-to-text, (2004), pp. 69–71. https://academic.microsoft.com/paper/97072897/reference
77. Kaldi Help Google Group: Multiple output heads in chain network (2020). https://groups.google.com/d/msg/kaldi-help/WC8hYgL2o3I/WccCc0ucAgAJ. Accessed 23 Mar 2020
78. A. Zeyer, K. Irie, R. Schlüter, H. Ney, in *Interspeech*. Improved training of end-to-end attention models for speech recognition, (2018), pp. 7–11. https://doi.org/10.21437/Interspeech.2018-1616
79. R. Sennrich, B. Haddow, A. Birch, in *2016 Meeting of the Association for Computational Linguistics*. Neural machine translation of rare words with subword units, (2016), pp. 1715–1725. https://doi.org/10.18653/v1/p16-1162
80. N. Zeghidour, N. Usunier, G. Synnaeve, R. Collobert, E. Dupoux, in *Interspeech*. End-to-end speech recognition from the raw waveform, (2018), pp. 781–785. https://doi.org/10.21437/Interspeech.2018-2414
81. T. Likhomanenko, G. Synnaeve, R. Collobert, in *Interspeech*. Who needs words? lexicon-free speech recognition, (2019), pp. 3915–3919. https://doi.org/10.21437/Interspeech.2019-3107
82. Wav2Letter lexicon-free LibriSpeech implementation (2020). https://github.com/facebookresearch/wav2letter/blob/master/recipes/models/lexicon_free/librispeech/am.arch. Accessed 23 Mar 2020
83. T. Salimans, D. P. Kingma, in *2016 Neural Information Processing Systems*. Weight normalization: A simple reparameterization to accelerate training of deep neural networks, (2016), pp. 901–909. https://academic.microsoft.com/paper/2963685250/reference
84. B. Ginsburg, P. Castonguay, O. Hrinchuk, O. Kuchaiev, V. Lavrukhin, R. Leary, J. Li, H. Nguyen, J. M. Cohen, Stochastic gradient methods with layer-wise adaptive moments for training of deep networks. arXiv preprint arXiv:1905.11286 (2019)
85. S. Kriman, S. Beliaev, B. Ginsburg, J. Huang, O. Kuchaiev, V. Lavrukhin, R. Leary, J. Li, Y. Zhang, in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Quartznet: Deep automatic speech recognition with 1d time-channel separable convolutions, (2020), pp. 6124–6128. https://doi.org/10.1109/icassp40776.2020.9053889
86. Open Speech and Language Resources (2020). http://www.openslr.org/resources/11/4-gram.arpa.gz. Accessed 23 Mar 2020

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Submit your manuscript to a SpringerOpen journal and benefit from:

- ► Convenient online submission
- ► Rigorous peer review
- ► Open access: articles freely available online
- ► High visibility within the field
- ► Retaining the copyright to your article

Submit your next manuscript at ► springeropen.com