THE UNIVERSITY
of EDINBURGH

Deep Representation Learning for Speech Recognition

Joanna Równicka

Doctor of Philosophy
Institute for Language, Cognition and Computation
School of Informatics
University of Edinburgh
2020

Abstract

Representation learning is a fundamental ingredient of deep learning. However, learning a good representation is a challenging task. For speech recognition, such a representation should contain the information needed to perform well in this task. A robust representation should also be reusable, hence it should capture the structure of the data. Interpretability is another desired characteristic. In this thesis we strive to learn an optimal deep representation for speech recognition using feed-forward Neural Networks (NNs) with different connectivity patterns.

First and foremost, we aim to improve the robustness of the acoustic models. We use attribute-aware and adaptive training strategies to model the underlying factors of variation related to the speakers and the acoustic conditions. We focus on low-latency and real-time decoding scenarios. We explore different utterance summaries (referred to as utterance embeddings), capturing various sources of speech variability, and we seek to optimise speaker adaptive training (SAT) with control networks acting on the embeddings. We also propose a multi-scale CNN layer, to learn factorised representations. The proposed multi-scale approach also tackles the computational and memory efficiency.

We also present a number of different approaches as an attempt to better understand learned representations. First, with a controlled design, we aim to assess the role of individual components of deep CNN acoustic models. Next, with saliency maps, we evaluate the importance of each input feature with respect to the classification criterion. Then, we propose to evaluate layer-wise and model-wise learned representations in different diagnostic verification tasks (speaker and acoustic condition verification). We propose a deep CNN model as the embedding extractor, merging the information learned at different layers in the network. Similarly, we perform the analyses for the embeddings used in SAT-DNNs to gain more insight. For the multi-scale models, we also show how to compare learned representations (and assess their robustness) with a metric invariant to affine transformations.

Acknowledgements

I would like to thank

- Steve Renals, for giving me the opportunity to pursue a PhD at the Centre for Speech Technology Research (CSTR) at the University of Edinburgh, for his guidance, enthusiasm, understanding, patience, encouragement to put my ideas into test, and for a scientific freedom.

- Peter Bell, for his expert advice, insightful suggestions, inspiring discussions, and priceless feedback at all of the stages of my research journey.

- My examiners: Hiroshi Shimodaira and Mathew Magimai Doss for peer-reviewing this thesis, for interesting discussions at the viva voce, and for providing a feedback which undoubtedly helped to improve this work.

- All CSTR members, for creating an extraordinary academic environment. I am grateful that I could be a member of this group. I would especially like to thank Ondrej Klejch for his helping hand from my day one, and for giving me the motivation to get up early to win the first-in-the-office competition. Special thanks also go to Joachim Fainberg for all the interesting conversations, and to Sameer Bansal, for his interest and feedback on my research. I would also like to thank Erfan Loweimi for the discussions that helped me to see my work in a different light. Also, thank you to Korin Richmond for providing a valuable feedback during annual reviews.

- My industry partners: the DataLab, Quorate, Ericsson and RedBee Media. I appreciate the opportunity to gain an industrial experience during my internships. I am also grateful for the financial support.

- Xue Li for being a precious friend. Thank you for your kindness, empathy, and support. This journey would have been much harder if I had not met you in Firbush.

- My family. Mateusz, thank you for your constant support and for believing in me. My mum, my dad, my sister. *Dziękuję Wam za wsparcie, rozmowy, zrozumienie. Bez Was nie dałabym rady.*

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Joanna Równicka*)

"Rabbit's clever," said Pooh thoughtfully.
"Yes," said Piglet, "Rabbit's clever."
"And he has Brain."
"Yes," said Piglet, "Rabbit has Brain."
There was a long silence.
"I suppose," said Pooh, "that that's why he never understands anything."

— A. A. Milne *Winnie-the-Pooh*

Acronyms

AI Artificial Intelligence.

AM Acoustic Modelling.

AMI Augmented Multi-party Interaction.

ASR Automatic Speech Recognition.

ASV Automatic Speaker Verification.

BSV Bottleneck Speaker Vector.

CAT Cluster Adaptive Training.

CD Context Dependent.

CE Cross-Entropy.

CI Context Independent.

CMN Cepstral Mean Normalisation.

CMVN Cepstral Mean and Variance Normalisation.

CNN Convolutional Neural Network.

DCT Discrete Cosine Transform.

DFT Discrete Fourier Transform.

DL Deep Learning.

DLN Dynamic Layer Normalisation.

DNN Deep Neural Network.

EER Equal Error Rate.

FBANK Mel-scaled Filterbank.

FC Fully-connected.

fDLR Feature-based Discriminative Linear Regression.

FFT Fast Fourier Transform.

FHL Factorised Hidden Layer.

fMLLR Feature space Maximum Likelihood Linear Regression.

GMM Gaussian Mixture Model.

HMM Hidden Markov Model.

IHM Individual Headset Microphone.

KL Kullback-Leibler.

LDA Linear Discriminant Analysis.

LF-MMI Lattice-Free MMI.

LHN Linear Hidden Network.

LHUC Learning Hidden Unit Contribution.

LIN Linear Input Network.

LM Language Model.

LN Layer Normalisation.

LON Linear Output Network.

LSTM Long Short-Term Memory.

LVCSR Large Vocabulary Continuous Speech Recognition.

MAP Maximum A Posteriori.

MDM Multiple Distant Microphones.

MFCC Mel-Frequency Cepstral Coefficients.

MGB Multi-Genre Broadcast.

MHSA Multi-Head Self-Attention.

ML Maximum Likelihood.

ML Machine Learning.

MLLT Maximum Likelihood Linear Transform.

MMI Maximum Mutual Information.

MPE Minimum Phone Error.

MSE Mean-Squared Error.

MT Machine Translation.

NaT Noise-aware Training.

NIST National Institute of Standards and Technology.

NLP Natural Language Processing.

NLU Natural Language Understanding.

NN Neural Network.

PCA Principal Component Analysis.

PDF Probability Density Function.

PER Phone Error Rate.

PLDA Probabilistic Linear Discriminant Analysis.

ReLU Rectified Linear Unit.

RIR Room Impulse Response.

RNN Recurrent Neural Network.

SA Speaker Adaptive.

SAT Speaker Adaptive Training.

SCTK Speech Recognition Scoring Toolkit.

SD Speaker Dependent.

SDM Single Distant Microphone.

SGD Stochastic Gradient Descent.

SI Speaker Independent.

SNR Signal-to-Noise Ratio.

SV Speaker Verification.

t-SNE t-distributed Stochastic Neighbour Embedding.

TDNN Time-Delay Neural Network.

UBM Universal Background Model.

WER Word Error Rate.

Notation

$\boldsymbol{x}_t$	input feature vector at time t
$\boldsymbol{X}$	sequence of input feature vectors over an utterance
$\hat{\boldsymbol{X}}_t$	sequence of input feature vectors over a window at time t
q_t	HMM hidden state at time t
$\boldsymbol{Q}$	sequence of hidden HMM states
$\boldsymbol{w}$	sequence of words
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$
θ	arbitrary set of parameters
$\mathcal{F}(\theta)$	objective function to be optimised over θ
$\boldsymbol{W}$	weight matrix
$\boldsymbol{b}$	bias vector
$\boldsymbol{e}_t$	embedding vector at time t
$\phi(\cdot)$	auxiliary network
$\boldsymbol{a}_t$	activation vector at time t
$\sigma(\cdot)$ or $\psi(\cdot)$	activation function

$\Delta(u)$	weight change at update time u
X	input tensor
Y	output tensor
$\boldsymbol{A}$	arbitrary matrix
$\boldsymbol{A}^T$	transpose of $\boldsymbol{A}$
$a_{i,j}$	element i, j of matrix $\boldsymbol{A}$

Contents

CHAPTER 1

Introduction

1.1 Motivation

Language, as defined by Collins English Dictionary, is "a system of communication which consists of a set of sounds and written symbols which are used by the people of a particular country or region for talking or writing". Language is used for inter-personal communication. However, technology plays an increasingly important role in our everyday life, hence developing a suitable communication system between humans and computers is desirable.

To enable human-computer communication via speech, Automatic Speech Recognition (ASR) technology is used to imitate the listening process. ASR is used as a component in virtual personal assistants (e.g. Apple's Siri, Google Assistant, Amazon Alexa), to enable human-computer communication. Those technologies have become omnipresent in recent years and their accuracy is constantly increasing, mainly due to the availability of specialised hardware (GPUs and TPUs) and large training datasets.

The goal of ASR systems is to transform speech into written text, i.e. provide an automatic transcription of speech. It can find application in dictation, e.g. for medical, political or forensic purposes. The transcription can also be used in other downstream tasks, such as translation, automatic closed captioning, Interactive

Voice Response (IVR), language or pronunciation learning. This broad spectrum of applications of speech recognition technology drives the interest into designing state-of-the-art models by both academia and industry.

Large scale Machine Learning (ML), i.e. automatically learning from large data quantities, has its source in speech recognition approaches from the 1970's and 1980's [Jelinek, 1976; Baker, 1975]. ASR is a hard ML problem. Both input and output spaces are sequential. The input signal is also highly variable, due to different recording devices, noise, reverberation, and speaker characteristics.

Data-driven ML approaches seek to train an accurate model of the speech-to-text process, i.e. to find a good mapping function $f : \boldsymbol{X} \rightarrow Y$, where $\boldsymbol{X}$ is a sequence of input vectors $\boldsymbol{x}$ and Y is a set of output classes y. Let the probability of class y given $\boldsymbol{x}$ be $P(y|\boldsymbol{x})$. The most likely output label y^* is given by

$$y^* = \underset{y \in Y}{\operatorname{argmax}} P_\theta(y|\boldsymbol{x}) \tag{1.1}$$

where $P_\theta(y|\boldsymbol{x})$ is the estimate of $P(y|\boldsymbol{x})$ and corresponds to the mapping function f parametrised by θ. Using Bayes' theorem, the equation can be rewritten as

$$y^* = \underset{y \in Y}{\operatorname{argmax}} p_\theta(\boldsymbol{x}|y)P(y). \tag{1.2}$$

The above equation is the underlying framework for Hidden Markov Model (HMM) based ASR systems [Baker, 1975]. In hybrid HMM-DNN systems, introduced by [Bourlard and Morgan, 1989], $p_\theta(\boldsymbol{x}|y)$ corresponds to the acoustic model, and $P(y)$ is a statistical language model. In supervised ASR, first, the input feature vectors $\boldsymbol{x}$ (representing the acoustics) and the labels y (representing the transcript) form training pairs and are used to automatically determine parameters θ of the acoustic model. Then, the most likely transcription can be determined at inference (i.e. decoding or test) time, when the model is presented with new data, unseen at training.

In this thesis we focus on learning representations to model the acoustics for speech recognition. To this end, we use Neural Networks (NNs), which are composed of multiple non-linear transformations yielding successively more abstract and richer features. We treat all hidden layers of a NN model as a speech representation. Neural Networks have been successfully used for acoustic modelling, giving state-of-the-art results in many speech recognition benchmarks [Hinton et al., 2012; Woodland et al., 2015; Saon et al., 2017]. Different deep architectures

have been previously explored – feed-forward (e.g. Deep Neural Network (DNN), Time-Delay Neural Network (TDNN), Convolutional Neural Network (CNN)), and recurrent (e.g. Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM) network).

In this thesis we use feed-forward NN architectures, with the focus on CNNs. Compared to the recurrent architectures, feed-forward models can achieve lower latency, therefore are applicable to real-time inference. Moreover, the performance of modern feed-forward architectures, such as deep Convolutional Neural Networks (CNNs), is competitive with the recurrent ones [Sercu and Goel, 2016a; Qian and Woodland, 2017; Zeghidour et al., 2018b]. Also, CNNs are state-of-the-art models in other ML areas, such as Computer Vision [Simonyan and Zisserman, 2015; Szegedy et al., 2015; He et al., 2016].

One of the main challenges in any deep learning application is generalisation. This is also true in ASR. The models can struggle to generalise beyond the data they have seen during training, i.e. to provide a representation useful at inference. The high variability of the speech signal contributes to the data distribution mismatch. It can be caused by many factors that are related to the speaker (e.g. voice pitch, accent, emotions), to the environment (e.g. noise, reverberation), or to the recording device (e.g. channel characteristics). A *robust* acoustic model should perform as well as possible on the data drawn from the distribution not seen during training. The motivation for training robust acoustic models stems from the interest to achieve a universally applicable model, with limited training data. Acoustic model adaptation to testing conditions can be used to improve the robustness of an ASR system, and it is the main focus of this thesis.

Alongside robustness, *efficiency* is also an important aspect of modern ASR systems and is of great practical value. Smaller models can be deployed on mobile devices, faster training saves compute time, and faster decoding is well suited for real-time ASR. Hence, improving the robustness alongside efficiency is a challenging, but desirable outcome and we also address it in this thesis.

Finally, we are also interested in the reasons behind deep CNN's good ASR performance. We analyse learned representations to improve their *interpretability*. We believe that this path of research can contribute to designing more accurate acoustic models, and can help to interpret and trust the predictions of NN-based statistical models.

We state five desiderata employed by the methods proposed in this thesis:

1. **The model accounts for high variability of speech.** On one hand, the model needs high capacity to model speech; all of the acoustic models in this thesis are NN-based. On the other hand, it has to avoid overfitting and be able to generalise well to testing conditions, i.e. take into account high variability of speech. To address this problem, we explore deep CNNs in Chapter 6, embeddings[1] for acoustic model adaptation in Chapter 4, and multi-scale representations in Chapter 5.

2. **The model can quickly adapt to testing conditions.** The generalisation performance can also be measured by the latency of adaptation at run-time. Some of the proposed methods are frame-wise, therefore allow for streaming adaptation and decoding, others are utterance-wise, therefore introduce a latency equal to the duration of the utterance at test time. We focus on real-time and low-latency decoding scenarios in this thesis.

3. **A disjoint adaptation dataset is not available.** With universality in mind, we do not allow for the use of data coming from the distribution of the testing conditions to adapt the model before inference, to avoid manual task-dependent tuning. The assumption is that we can not predict the conditions in which the model will be deployed. In our approaches, we model the variability of the speech signal naturally provided at training (by using multi-condition training sets) and at inference.

4. **Adaptation is unsupervised.** The methods developed in the thesis do not make use of any transcriptions for adaptation, which is a desired feature on its own, but is also a product of desiderata 2 (i.e. using hypotheses from the first-pass decode as labels in a semi-supervised setup would increase the latency at inference) and 3 (i.e. we do not use a disjoint adaptation set which could be provided with a manual transcription). Some of the methods make the use of the information about the identity of the speakers at test time, but the adaptation is always performed more frequently than per speaker, to allow fast adaptation.

5. **Supervised training data and computational resources are limited.** With unlimited computational resources and training data covering all pos-

[1]Throughout this thesis, an embedding is a learned low-dimensional vector representation of a speech frame, an utterance, a speaker, or an acoustic condition.

sible sources of variability, $p_\theta(\boldsymbol{x}|y)$ should be close to the true distribution $p(\boldsymbol{x}|y)$. Some of the recent end-to-end approaches head in this direction – they can leverage huge amounts of training data to improve the recognition. However, in this thesis we focus on small and medium size ASR tasks, thus the acoustic models are trained in a hybrid DNN-HMM framework, in a supervised manner. We assume a limited training set of limited variability and limited computational resources.

1.2 Research objectives and contributions

Our goal is to learn good representation of the input data describing the acoustics with a multi-layer NN. An optimal deep representation for speech recognition should

- enable low classification error,
- capture the underlying structure of the data,
- be interpretable.

To satisfy those requirements, the representation should be *disentangled*. Bengio et al. [2013] links the disentanglement to the sparsity of the representation over the transformations in the data. This implies a factorised representation, which separates latent causes of variation in the data. A disentangled representation captures the underlying structure of the data, hence it improves the robustness to out-of-distribution testing conditions.

Some of the underlying factors of variation in a speech signal are informative (they help to reduce the classification error), while some are uninformative, and can worsen the generalisation performance. Input to the acoustic model can be composed of the interaction between one or more audio sources (e.g. vocal tract of a speaker and a source of background noise) and of the interaction of local time-frequency patterns, which differentiate speech sounds, but are also a reflection of the variability in audio sources (i.e. the variability caused by different types of noises or speaker's accent, emotions etc.). All factors of variation contribute to the input representation, and modelling the informative ones while reducing the sensitivity to directions of uninformative variance is a challenging task.

Disentanglement is closely related to interpretability. An interpretable representation can be easily understood in human terms, and such a representation is typically disentangled, in a sense that it expresses the generative structure of the data. A better understanding of learned representations can lead to designing more accurate models.

The objective of this thesis is to improve deep representation learning for speech recognition. We hypothesise that the disentanglement of the informative factors of variation can lead to an optimal representation. To improve interpretability, we also test hidden representations in verification tasks, to better understand if and how the underlying factors of variation are reflected in learned representations, indicating the usefulness of those factors for robust ASR.

In terms of the contributions of the thesis,

- we propose novel utterance summary vectors (referred to as utterance embeddings) for acoustic model adaptation, to learn disentangled representations by providing the model with the information about the underlying factors of variation changing at the utterance level (Chapter 4),
- we provide recommendations for the best way to incorporate the embeddings into the acoustic models, and we compare our embeddings with other state-of-the-art embeddings in attribute-aware and adaptive training frameworks (Chapter 4),
- we propose a multi-scale convolutional layer to learn robust factorised representations, while reducing the computational cost and memory footprint (Chapter 5),
- we propose a deep CNN architecture, visualisations of learned representations, diagnostic verification, and a comparison of learned representations to better understand the representations learned by the acoustic models (Chapter 6).

The work in this thesis primarily relates to the following papers:

1. **Rownicka, J.**, Renals, S., and Bell, P. (2017). Simplifying very deep convolutional neural network architectures for robust speech recognition. In *Proc. IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*.

2. **Rownicka, J.**, Bell, P., and Renals, S. (2018). Analyzing deep CNN-based utterance embeddings for acoustic model adaptation. In *Proc. IEEE Spoken Language Technology Workshop (SLT)*.

3. **Rownicka, J.**, Bell, P., and Renals, S. (2019). Embeddings for DNN speaker adaptive training. In *Proc. IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*.

4. **Rownicka, J.**, Bell, P., and Renals, S. (2020). Multi-scale octave convolutions for robust speech recognition. In *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

The following papers are also related and were published during the course of my studies, but are not discussed in details in the thesis:

5. Williams, J., and **Rownicka, J.** (2019). Speech replay detection with x-vector attack embeddings and spectral features. In *Proc. Interspeech 2019*.

6. Williams, J., **Rownicka, J.**, Oplustil, P., and King, S. (2020). Comparison of speech representations for automatic quality estimation in multi-speaker text-to-speech synthesis. In *Proc. Odyssey 2020 The Speaker and Language Recognition Workshop*.

7. Oplustil, P., Williams, J., **Rownicka, J.**, and King, S. (2020). An unsupervised method to select a speaker subset from large multi-speaker speech synthesis datasets. In *Proc. Interspeech 2020*.

CHAPTER 2

Neural Networks for Automatic Speech Recognition

This chapter gives an overview of Automatic Speech Recognition (ASR) and Deep Learning (DL) techniques used for acoustic modelling.

2.1 Overview of Automatic Speech Recognition

The goal of speech recognition is to transform an acoustic waveform into a string of words $\boldsymbol{w} = (w_1, w_2, ..., w_N)$. Under the probabilistic framework, this task can be formulated as a search for the most likely sequence of words $\boldsymbol{w}^*$, out of all possible word sequences, given some acoustic input $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_T)$, where $N << T$. The acoustic input representations are described in Section 2.1.2. The objective of the statistical ASR is to find a sequence of words which maximises the posterior probability of a sentence given the sequence of acoustic feature vectors:

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmax}}\, P(\boldsymbol{w}|\boldsymbol{X}) \tag{2.1}$$

Directly estimating the posterior probability of a sentence from the acoustics underlies end-to-end ASR systems, described briefly in Section 2.2.5. In this thesis, however, we use Bayes' rule to find the most likely sequence of words. The

above equation can be decomposed as

$$\begin{aligned} \boldsymbol{w}^* &= \underset{\boldsymbol{w}}{\operatorname{argmax}} \frac{p(\boldsymbol{X}|\boldsymbol{w})P(\boldsymbol{w})}{p(\boldsymbol{X})} \\ &= \underset{\boldsymbol{w}}{\operatorname{argmax}}\, p(\boldsymbol{X}|\boldsymbol{w})P(\boldsymbol{w}). \end{aligned} \tag{2.2}$$

The probability of the acoustic observation sequence, $p(\boldsymbol{X})$, can be ignored, since it is independent of the word sequence. $p(\boldsymbol{X}|\boldsymbol{w})$ corresponds to the observation likelihood, and is computed with an acoustic model, which is the focus of this thesis. In Sections 2.1.3-2.1.5 we give an overview of the approaches to model and estimate this likelihood. P($\boldsymbol{w}$) is the prior probability of the sequence of words, and can be estimated with an n-gram language model (described in Section 2.1.6) or a NN-based language model. A lexicon, i.e. a list of words with pronunciations expressed as a phone sequence, is a third component needed for ASR in a hybrid framework.

To perform the recognition after the model is trained, we combine the acoustic model (a sequence of acoustic likelihoods), a lexicon, and a language model (an n-gram grammar), in order to obtain the most likely sequence of words. This is decoding phase and it is described in more detail in Section 2.1.7. First, we describe the metric used for the evaluation of ASR systems.

2.1.1 Evaluation

The primary evaluation metric for ASR systems is Word Error Rate (WER). It indicates how much the hypothesised word sequence differs from the reference word sequence. String distance enables to determine minimum edit distance alignment in words[1], i.e. the minimum number of edits to transform one sequence into the other. The algorithms for finding minimum edit distance are based on dynamic programming [Bellman, 1957]. Different programs for finding the alignment can have different hyper-parameters, which can affect the alignment and the error computation.

Three types of edits are typically distinguished: insertions (I), deletions (D), and substitutions (S). For N words in the reference transcript, the WER is defined

[1]Given reference phone annotation, minimum edit distance at the phone level can be used to compute Phone Error Rate (PER).

as:

$$WER = \frac{I + D + S}{N} \times 100\% \tag{2.3}$$

WER is typically used for comparisons between different models and approaches, since ASR experiments are often performed on benchmark corpora. We test our methods on multiple corpora to show the validity of the proposed approaches. In the thesis, we do not include the WERs for the models with different seeds used for initial weights sampling. We tested our best performing models from Chapters 4, 5 and 6 with three different seeds and we did not observe significant differences in WERs between the runs.

We also perform significance tests for our main results, especially when the differences between WERs are small. We discuss the significance tests results in Conclusions of Chapters 4, 5 and 6. Significance testing for ASR requires to take into account the independence of the errors. Gillick and Cox [1989] proposed a matched-pairs test to evaluate ASR systems. The test can be used when errors are not independent events. We use SCLITE's Statistical System Comparison Program (sc_stats) from Speech Recognition Scoring Toolkit (SCTK) from National Institute of Standards and Technology (NIST) to compare our methods. The null hypothesis is that there is no performance difference between the systems. We report the minimum value of p for which the test finds a significant difference at the level of p. A p-value higher than 0.05 is considered not statistically significant.

2.1.2 Input representations

Acoustic waveform is a digitised and quantised analog speech waveform, showing the amplitude measurements of the signal at a particular time. A waveform signal has high data rate, with the number of samples per second dependent on the sampling frequency. To provide a more compact representation for ASR systems, many hand-crafted features have been developed over the years to facilitate phone discrimination. One of the most successful ones are Mel-Frequency Cepstral Coefficients (MFCC), introduced by Bridle and Brown [1974] and Davis and Mermelstein [1980] in the 1980's. Their design is inspired by human auditory system physiology; they are used by many modern ASR systems to this day.

Figure 2.1 shows the extraction diagram for MFCC and other acoustic features used as front-end representations for our deep learning models. A first step of

MFCC extraction is pre-emphasis, i.e. first-order high-pass filtering, to boost the amount of energy in the high frequencies. Dithering [Borsky et al., 2017] (white noise addition to reduce distortion of low-amplitude signals) and DC offset removal (centring the signal) are also usually applied at this stage.

The assumption for the acoustic feature extraction is that speech signal is piece-wise stationary. To obtain good time resolution, the signal can therefore be divided into short frames (typically 25ms with 10ms overlap) and extracted with a window (typically based on a Hamming window). A windowed signal at time step t can be used to compute energy, as the sum over time of the power of the samples[2] and appended to the feature vector later in the process.

Next, the time-domain signal is converted into frequency domain by Discrete Fourier Transform (DFT), with Fast Fourier Transform (FFT) algorithm. The magnitude of the frequency components at time step t is a spectrum, and *spectrogram features* are formed by stacking several consecutive spectrum feature vectors.

To model unequal sensitivity of human hearing at all frequency bands, the mel-filter bank, consisting of 20-80 overlapping triangular band-pass filters for a good frequency resolution, is then applied to the power spectra, and the filter-bank energies (i.e. mel-filter bank values) are then compressed with a logarithm. The feature vectors at this point are referred to as *FBANK features*, and are used in this thesis for most of the experiments. Mean or mean and variance normalisation (MN/MVN) can be applied to FBANKs, to make them more suitable for further processing with neural networks [LeCun et al., 1998b]. The statistics for the normalisation can be estimated offline per recording (per utterance) or per speaker, or as a moving average for real-time ASR.

For MFCCs, a Discrete Cosine Transform (DCT) (equivalent to inverse DFT) is used to decorrelate mel filter-bank energies, resulting in cepstral coefficients. Decorrelation property is the most important for the HMM-GMM systems with a diagonal covariance matrix. It is not required for discriminatively trained hybrid NN-HMM systems, however, it can be a desirable property [LeCun et al., 1998b].

Traditionally, not all of the DCT coefficients were kept. By discarding higher values in cepstral domain one could remove the glottal source characteristics (i.e. the fundamental frequency), which are not as relevant for phone discrimination as the vocal tract filter information [Jurafsky and Martin, 2009, Chapter 9]. More

[2] Alternatively, the average of log energy can be used instead of energy.

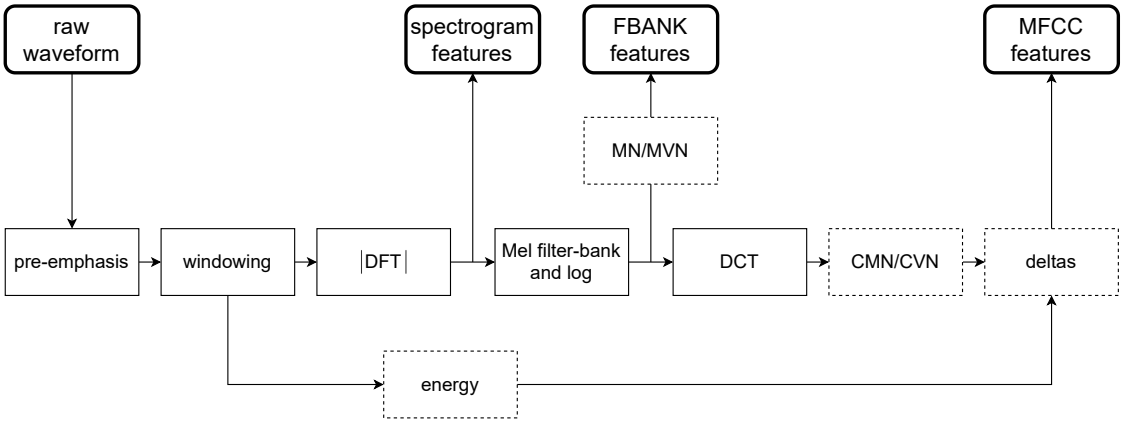


Figure 2.1: Extraction steps for input features representations. Optional steps are denoted with a doted line.

recently, retaining all of the coefficients proved to be effective for ASR [Peddinti et al., 2015b], suggesting that the characteristics of the glottal source might be useful in this task.

Cepstral Mean Normalisation (CMN) or Cepstral Mean and Variance Normalisation (CMVN) can be applied similarly to mean and variance normalisation described earlier, but in cepstral domain this transformation compensates for the effects of unknown linear filtering, making the representations more robust to channel variations [Liu et al., 1993]. The normalisation in cepstral domain has also proved to be effective in noisy conditions [Viikki and Laurila, 1998]. Finally, dynamic features, i.e. delta and double delta coefficients can be appended [Furui, 1986], to model how the cepstral coefficients change over time. Dynamic features may also help to compensate for the conditional independence assumption of the observations (see Section 2.1.3)[3].

In most of our experiments, we use normalised FBANK features as input representations. They are correlated, however, this can be a desired property when used as the input to a Convolutional Neural Network (CNN) with two-dimensional kernels, which was designed to exploit local spatial correlations of the data. We use such architectures in many of our experiments.

Spectrogram features show how the frequencies change over time, before filter-bank application. Spectrogram features are therefore more correlated but less compressed than FBANKs, hence they might contain more details differentiating

[3]Alternatively, Linear Discriminant Analysis (LDA) transform and Maximum Likelihood Linear Transform (MLLT) [Gales, 1999] can be applied to the input features spliced across several frames.

sounds than the other representations. Also, not applying a filter bank will not smooth the spectrum, which can prevent hindering the extraction of pitch and formants. Those characteristics might potentially be useful in input representations, for disentangling non-linguistic variability factors by a NN-based acoustic model. We use spectrogram features for multi-scale representation learning in Chapter 5.

It is also worth mentioning that some of the recent ASR systems have proposed feeding the NN directly with the raw waveform. The premise is that the network can implicitly learn optimal transformations of the raw time-domain input, without the need for hand-crafted input representations. Usually a convolutional front-end is used in conjunction with raw waveform input, as e.g. by Palaz et al. [2015]; Golik et al. [2015]; Ghahremani et al. [2016a], with the objective to learn the filters and improve the accuracy. This goal turned out to be difficult to achieve[4].

To boost the performance of raw waveform-based ASR systems, injecting the prior knowledge about the auditory system was proposed, by initialising the weights of the first convolutional layer. Non-parametric Gammatone filters or parametric perceptual scales were used previously for this purpose.

Sainath et al. [2015c] used Gammatone filters for initialisation. They showed that the performance of raw waveform input can match FBANK input, with a model with time and frequency convolution frond-end, trained on 2 000 hours of data. However, the FBANK features were still complementary to the raw input in this work, suggesting that manually designing the features is still beneficial; the network was not able to automatically extract optimal representations.

SincNet [Ravanelli and Bengio, 2018] is an example of a parametric method to inject the prior knowledge on the filter shape. Cut-off frequencies for a bank of rectangular band-pass filters are the only parameters learned, hence the number of parameters is substantially reduced, compared to a vanilla CNN model. Loweimi et al. [2019] investigated the replacement of the rectangular filters with triangular, Gammatone and Gaussian filters, and showed PER reduction. Moreover, the authors explore the representations learned by the models, and find that the filters of the first layer are similar to mel-filter bank used in standard input representations extraction. Parcollet et al. [2020] use SincNet for end-to-end ASR and show the gains in performance compared to a vanilla CNN model.

[4]Interestingly, the learned filters often resembled the hand-designed ones.

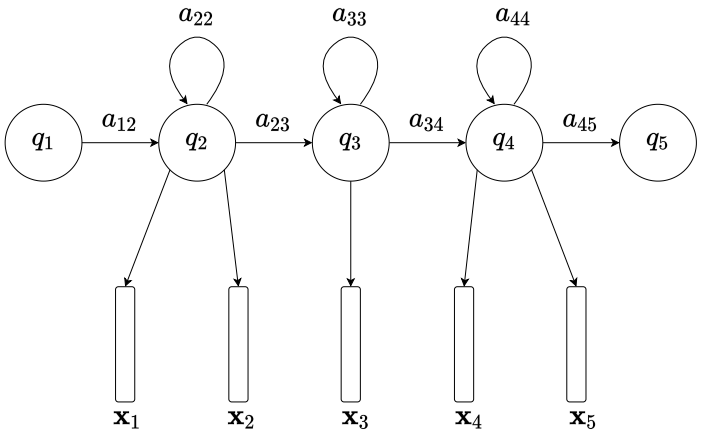


Figure 2.2: An example of an HMM model with 5 states. States q_1 and q_5 are non-emitting, i.e. not associated with the observations. The observation vectors are denoted with $\boldsymbol{x}_1, ..., \boldsymbol{x}_5$.

In summary, employing the knowledge about the physiology of speech and hearing into the overall model, at the input or hidden representation level, is still crucial for optimal ASR performance. In our experiments, we train our acoustic models on FBANKs and spectrogram features.

2.1.3 Hidden Markov acoustic models

In this thesis, we are concerned with Hidden Markov Model (HMM) based acoustic models. In this section, we briefly introduce how the HMMs are used for ASR. For a more complete introduction, we refer the reader to [Gales and Young, 2007].

The HMM is a generative model for the sequence of observations. In training the HMM-based ASR systems, the HMM provides a way to align the input and output sequences, $\boldsymbol{X}$ and $\boldsymbol{w}$. In decoding, it enables to search over all possible sequences $\boldsymbol{w}$, to find the most likely one $\boldsymbol{w}^*$. Figure 2.2 depicts an example of an HMM model applied to speech.

At each time step t, the HMM model is in state q_t and it generates an observation $\boldsymbol{x}_t$. A model of N states is characterised by the following components:

- a set of discrete states $J = \{1, ..., N\}$ (indexed by i or j),
- a transition probability matrix $\boldsymbol{A} = \{a_{ij}\}$, specifying the probability of moving from state i to state j as

$$a_{ij} = P(q_t = j | q_{t-1} = i) \tag{2.4}$$

$$\sum_{j=1}^{n} a_{ij} = 1, \forall i = 1, ..., N, \tag{2.5}$$

- a set of observation likelihoods (i.e. emission probabilities), $\boldsymbol{B} = \{b_j(\boldsymbol{x}_t)\}$, expressing the probability of an observation vector $\boldsymbol{x}_t$ being generated from state j at time t as

$$b_j(\boldsymbol{x}_t) = p(\boldsymbol{x}_t|q_t = j). \tag{2.6}$$

Most commonly, a Probability Density Function (PDF) is used to model emission probability distributions $\boldsymbol{B}$. In Section 2.1.5 we briefly describe the models used for acoustic likelihood computation – a Gaussian Mixture Model (GMM) and a Neural Network (NN). In fact, a Neural Network estimates scaled likelihood $\frac{p(\boldsymbol{x}_t|q_t=j)}{p(\boldsymbol{x}_t)}$, rather than the likelihood $p(\boldsymbol{x}_t|q_t = j)$. Since $p(\boldsymbol{x}_t)$ does not depend on q_t, scaled likelihoods can be used as output probabilities in a hybrid HMM-NN framework, instead of the likelihoods as in HMM-GMM models.

For speech modelling applications, the topology of the HMM is usually restricted by the first-order Markov assumption – the state q_t at time t is conditionally independent of all previous states but q_{t-1}:

$$P(q_t|q_{t-1}, q_{t-2}, ..., q_1) = P(q_t|q_{t-1}) \tag{2.7}$$

Also, the observation vectors $\boldsymbol{x}_t$ at time t are assumed to be conditionally independent of previous observation vectors and previous states, given q_t:

$$p(\boldsymbol{x}_t|\boldsymbol{X}_{1:t-1}, \boldsymbol{Q}_{1:t-1}) = p(\boldsymbol{x}_t|q_t) \tag{2.8}$$

$\boldsymbol{Q}$ denotes a state sequence, and $\boldsymbol{X}$ is a sequence of observations for previous time steps. The likelihood of the acoustics can therefore be computed as:

$$p(\boldsymbol{X}|\boldsymbol{w}; \theta) = \sum_{\boldsymbol{Q}(\boldsymbol{w})} \prod_{t=1}^{T} P(q_t|q_{t-1}; \theta) p(\boldsymbol{x}_t|q_t; \theta) \tag{2.9}$$

where $\theta = \{\boldsymbol{A}, \boldsymbol{B}\}$ is the set of parameters of the HMM model. All possible state sequences are denoted by $\boldsymbol{Q}(\boldsymbol{w})$. Naively computing a separate observation likelihood for each hidden state sequence and summing them is too complex for real-world tasks. Instead, an efficient *forward* algorithm is used to sum over a large number of state sequences.

To update the HMM parameters θ at training, the *forward-backward* algorithm (also known as Baum-Welch) is used [Baum and Eagon, 1967]. In decoding, the *Viterbi* algorithm [Viterbi, 1967] is used to determine the most likely sequence of states. The Viterbi algorithm can also be used as an efficient approximation to Baum-Welch training, where only the most probable state sequence is implemented, by replacing $\sum_{\boldsymbol{Q}(\boldsymbol{w})}$ with $\max_{\boldsymbol{Q}(\boldsymbol{w})}$ in Equation 2.9. Running the Viterbi algorithm on the training data, where the word sequence is known, is often referred to as forced alignment.

2.1.4 Acoustic units

For Large Vocabulary Continuous Speech Recognition (LVCSR) within the HMM paradigm, it is necessary to divide words into sub-words units before modelling. A basic speech sound is a phone (i.e. Context Independent (CI) phone, or a monophone). However, depending on the context of a phone, its acoustic realisation changes (the effect known as co-articulation). Therefore, usually Context Dependent (CD) phones are used as the units for HMM modelling. The phones with immediate left and right context are referred to as triphones [Schwartz et al., 1985].

Constructing HMM models for all possible triphones would require too many parameters to train. The most common approach to address this problem is to cluster some of the contexts and tie states whose contexts fall into the same cluster [Young and Woodland, 1994]. Tied states share the same parameters, hence the number of parameters is reduced. In the thesis we refer to context-dependent tied-states as senones [Mei-Yuh Hwang et al., 1996].

2.1.5 Acoustic likelihood computation

To train and decode with an HMM-based ASR system, an observation likelihood function is required to compute the likelihoods for continuous acoustic observations, such as MFCCs. In this section, we briefly introduce GMMs and NNs, in the context of this task.

HMM-GMM models

The distributions of the features in the acoustic observations $\boldsymbol{x}_t$ do not have to be normal. For this reason, a weighted mixture of multivariate Gaussians, i.e. a

Gaussian Mixture Model (GMM) is often used as a Probability Density Function (PDF) [Rabiner et al., 1985]. In the HMM-GMM approach, the output likelihood function for the state j is modelled with M_j Gaussians as

$$b_j(\boldsymbol{x}_t) = p(\boldsymbol{x}_t|q_t = j) = \sum_{m=1}^{M_j} c_{jm}\mathcal{N}(\boldsymbol{x}_t; \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm}). \tag{2.10}$$

c_{jm} are mixture weights. The Gaussian PDF for $\boldsymbol{x}_t \in \mathbb{R}^d$, for state j and mixture m, is defined as

$$\mathcal{N}(\boldsymbol{x}_t; \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm}) = \frac{1}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}_{jm}|}} \exp\{-\frac{1}{2}(\boldsymbol{x}_t - \boldsymbol{\mu}_{jm})^T\boldsymbol{\Sigma}_{jm}^{-1}(\boldsymbol{x}_t - \boldsymbol{\mu}_{jm})\}. \tag{2.11}$$

The parameters of the PDFs are estimated form the training set data for which the transcription $\boldsymbol{w}$ is provided. To fit the best model to the data, the objective for training is Maximum Likelihood (ML):

$$\mathcal{F}_{ML}(\theta) = \log p(\boldsymbol{X}|\boldsymbol{w}; \theta) \tag{2.12}$$

Therefore, in HMM-GMM models, the parameters to estimate are $\theta = \{\boldsymbol{A}, \{\boldsymbol{c}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\}\}$. They are utterance-specific; an average over all utterances is actually used for optimisation over the whole training set, using Baum-Welch algorithm.

An alternative to fitting the best model to the data is discriminative training. At the frame level, the classifier gives a posterior estimate over HMM states. This approach was first proposed by Bourlard and Morgan [1994]; it is often referred to as hybrid HMM-NN, and this thesis focuses on hybrid HMM-NN models.

Discrminative training can also be realised at the word sequence level. Bahl et al. [1986] first proposed to maximise the mutual information (MMI) between the competing models, and Woodland and Povey [2002] provided a practical implementation of MMI estimation, leading to improvements in WERs. In [Povey and Woodland, 2002], the authors also proposed an alternative sequence level objective function, Minimum Phone Error (MPE), which is directly related to phone accuracy and WER.

Hybrid HMM-NN models

Applying deep learning techniques to the speech recognition task often enables to obtain state-of-the-art acoustic models [Hinton et al., 2012]. A NN can replace

the GMM to model posterior probability over HMM states. Typically, a window of acoustic observations $\hat{\boldsymbol{X}}_{\boldsymbol{t}} = (\boldsymbol{x}_{t-c}, ..., \boldsymbol{x}_t, ..., \boldsymbol{x}_{t+c})$, with left and right context c, is used at each time step t. To use the posterior probability estimated by a NN as output probabilities in an HMM, Bayes' rule is applied to obtain scaled likelihoods, by scaling with the HMM states priors $p(j)$. The function for the state j becomes

$$b_j(\boldsymbol{x}_t) = p(\boldsymbol{x}_t|q_t = j) = \frac{p(j|\hat{\boldsymbol{X}}_t)}{p(j)}. \tag{2.13}$$

HMM state priors $p(j)$ are obtained from forced alignment. The posterior probability of state j given acoustic observations $\hat{\boldsymbol{X}}_t$ at time t, $p(j|\hat{\boldsymbol{X}}_t)$, can be estimated with a parametric function $f(\hat{\boldsymbol{X}}_t; \theta)$. Many different forms of this function have been proposed for ASR over the years. In this thesis we use DNNs, TDNNs, and CNNs. In Section 2.2.4 we give an overview of different types of deep learning architectures used for this purpose.

The training objective for a disriminative model is to discriminate the best model from all other models, rather that fit the best model to the data. In hybrid HMM-NN systems, the objective is to minimise the difference between two distributions – the one obtained from the HMM-GMM system, one-hot encoding of forced aligned senones $\boldsymbol{y}$, and the one predicted by the NN, $p(q|\hat{\boldsymbol{X}}; \theta)$. Kullback-Leibler (KL) divergence of model predictions from the targets can be used for this purpose. It is defined as

$$\begin{aligned} D_{KL}(\boldsymbol{y} \,||\, p(q|\hat{\boldsymbol{X}}; \theta)) &= \sum_{t=1}^{T} \boldsymbol{y}_t \log \frac{\boldsymbol{y}_t}{p(q_t|\hat{\boldsymbol{X}}_t; \theta)} \\ &= \sum_{t=1}^{T} \boldsymbol{y}_t \log p(\boldsymbol{y}_t) - \sum_{t=1}^{T} \boldsymbol{y}_t \log p(q_t|\hat{\boldsymbol{X}}_t; \theta). \end{aligned} \tag{2.14}$$

Since the first term in the above equation does not depend on θ, the training objective, known as Cross-Entropy (CE) loss function, becomes

$$\mathcal{F}_{CE}(\theta) = -\sum_{t=1}^{T} \boldsymbol{y}_t \log p(q_t|\hat{\boldsymbol{X}}_t; \theta). \tag{2.15}$$

Alternatively, sequence level discriminative loss function can be used. Povey et al. [2016] proposed Lattice-Free MMI (LF-MMI) method, to compute HMM state posteriors directly with a NN, without CE pre-training, and without the

need to compute denominator lattices. The WERs are typically lower when trained with LF-MMI, compared to CE training. One drawback of LF-MMI training is its sensitivity to unreliable transcripts.

This thesis concerns deep learning models to directly estimate posterior distribution over senones with CE loss function, given hard aligned targets from generative HMM-GMM systems[5].

2.1.6 Language model

The Language Model (LM) is a prior distribution of words, $P(\boldsymbol{w})$ in Equation 2.2, showing the probability that a sequence of words $\boldsymbol{w} = (w_1, w_2, ..., w_N)$ is a valid word sequence.

In this thesis, we use the language models based on n-gram grammars. Those models estimate the prior probability of a sentence as

$$P(\boldsymbol{w}) \approx \prod_{k=1}^{K} P(w_k | w_{k-1}, w_{k-2}, ..., w_{k-N+1}). \tag{2.16}$$

The probabilities are estimated by maximising the likelihood of training data. For higher values of N, this approach leads to sparse data and a large number of poorly estimated zero probability n-grams. Typically, smoothing is used to address this problem [Jurafsky and Martin, 2009, Chapter 4].

Also, state-of-the-art NN-based language models can be employed into the ASR pipeline to ease the data sparsity problem. Neural language models can generalise better than n-gram models; they can also capture long range dependencies. Nevertheless, neural language models are typically used in second-pass rescoring, as e.g. in [Deoras et al., 2011]. In the first pass, n-gram models are still used in hybrid ASR systems, due to their low computational cost, and thus high efficiency at inference.

2.1.7 Decoding

After training the acoustic and the language models, one can obtain the transcription for the utterances at test time according to Equation 2.2. In practice, log-probabilities are used instead of probabilities, to avoid numeric underflow.

[5] In some of the experiments we refine the GMM alignments with TDNN alignments, and use them for our CNN models. However, the initial alignment in this work is always GMM-based.

We use lattice-based decoding passes. A lattice is a representation of alternative likely transcriptions (with associated alignment and cost information). The Viterbi algorithm is used to generate the lattice. For efficient computations, beam search is employed to prune unlikely hypotheses.

In this thesis, we scale the acoustic model at decoding, to account for incorrect independence assumptions resulting in correlations between frames [Jurafsky and Martin, 2009]. We use acoustic scale $\alpha = 0.1$, while the language model weight $\beta = 1.0$. The goal of the decoder becomes

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmax}}\{\alpha \log p(\boldsymbol{X}|\boldsymbol{w};\theta) + \beta \log P(\boldsymbol{w})\}. \tag{2.17}$$

To get the best path during scoring, we re-score the lattices with the language model weight from 7 to 15, and we report the best WER.

2.2 Deep learning for acoustic modelling

This section introduces deep learning, its training principles, and regularisation procedures, as well as different approaches to supervision and network topology design for acoustic modelling.

A neural network is a mapping function $f(\hat{\boldsymbol{X}}_t;\theta)$ parameterised by $\theta = \{\boldsymbol{W}, \boldsymbol{b}\}$. It maps the input features $\hat{\boldsymbol{X}}_t$ to the training targets $\boldsymbol{y}_t$. Different training targets can be used; an overview of different approaches to supervision in ASR can be found in Section 2.2.3. In general, the goal of training is to find θ given a set of tuples $\{\hat{\boldsymbol{X}}_t, \boldsymbol{y}_t\}$. The algorithms for NN training are briefly described in Section 2.2.1.

A NN is a nested function of L functions, where L denotes the number of *hidden layers* in deep learning terminology. Each hidden layer consists of I vector-to-scalar functions, referred to as *hidden units*. Neural Networks can be also seen as computational graphs, where each node corresponds to an operation, and the data flows between the nodes. More complex operations can be formed from the simpler ones.

A core operation used by NNs is a linear transformation, i.e. a dot product of the input features and a weight matrix $\boldsymbol{W}$, and a shift by a bias vector $\boldsymbol{b}$. In Section 2.2.4 we define those operations with regard to different network topologies. In general, the output of a linear transformation at time t is an activation vector $\boldsymbol{a}_t$.

The linear transformation is usually followed by a non-linear operation, referred to as *activation function*. In this thesis, we mostly use Rectified Linear Unit (ReLU) [Nair and Hinton, 2010] in the hidden layers and softmax in the output layer, but also sigmoid and tanh for the transformations acting on the embeddings in Chapter 4.

- *Sigmoid* bounds its inputs to $[0, 1]$ as:

$$\sigma(\boldsymbol{a}_t) = \frac{1}{1 + \exp(-\boldsymbol{a}_t)} \tag{2.18}$$

  The output can therefore be interpreted as the probability of the incoming representation. However, the saturation of the sigmoid function leads to small derivatives and slow training. Moreover, the statistics of the sigmoid outputs are not desirable, since the outputs are not centred at 0.

- *Tanh* (hyperbolic tangent) is centred at zero, providing better numerical characteristics. Moreover, tanh outputs can be positive or negative. The function is defined as:

$$\sigma(\boldsymbol{a}_t) = \frac{\exp(\boldsymbol{a}_t) - \exp(-\boldsymbol{a}_t)}{\exp(\boldsymbol{a}_t) + \exp(-\boldsymbol{a}_t)} \tag{2.19}$$

- *ReLU* has proved to be superior to other non-linearities in many ASR tasks. It enforces sparsity in the activation space as:

$$\sigma(\boldsymbol{a}_t) = \max(0, \boldsymbol{a}_t) \tag{2.20}$$

  For ReLU, there is no positive saturation that could lead to small derivatives in the backward gradient descent pass, potentially contributing to more stable training. The function is also computationally efficient.

- *Softmax* [Bridle, 1990] is usually used for multi-class classification. As for the sigmoid activation function, the output of the non-linearity is bounded by $[0, 1]$. Moreover, softmax ensures that all C outputs sum to one. Softmax output can therefore be interpreted as an estimate of a probability distribution. It is defined as:

$$\sigma(\boldsymbol{a}_t) = \frac{\exp(\boldsymbol{a}_t)}{\sum_{c=1}^{C} \exp(\boldsymbol{a}_c)} \tag{2.21}$$

Other operations, such as dropout, batch or layer normalisation can also be applied. They are used for regularisation purposes and are briefly described in Section 2.2.2.

2.2.1 Training

Neural Network training consists in parameter optimisation, in most cases via *gradient descent* and *back-propagation*. The training objective is to minimise the *loss function* . In this thesis we use the CE loss function, defined previously in Equation 2.15. An average over all utterances is used in training as the final criterion.

Gradient descent is an iterative process of updating the parameters θ of the model, and the technique for computing the gradient is referred to as back-propagation. Given the order of operations in the forward pass, the gradient computation is performed automatically by modern deep learning toolkits, such as Tensorflow [Abadi et al., 2015] or PyTorch [Paszke et al., 2019]. We take advantage of this feature in most of our implementations.

In Stochastic Gradient Descent (SGD), the parameters are updated after each training example. Alternatively, a mini-batch of training examples can be used to do the update. We follow this approach in this thesis. The steps for training a neural network are:

1. Randomly initialise parameters $\theta = \{\boldsymbol{W}, \boldsymbol{b}\}$, using an initialisation scheme[6].
2. Randomise the order of T training examples $\hat{\boldsymbol{X}}$ and divide them into mini-batches of size B.
3. For each mini-batch:
   - Compute the output for the examples $b = 1, ..., B$ and the average loss $\frac{1}{B}\sum_b^B \mathcal{F}_{CE}(\hat{\boldsymbol{X}}_b; \theta)$ with a forward pass.
   - Using the chain rule of differentiation, compute the gradients of the loss with respect to θ for each example, and accumulate the gradients

[6]To keep the variance of each layer constant, we either use Glorot and Bengio's initialisation for the weights $\boldsymbol{W}$ [Glorot and Bengio, 2010], which draws from uniform distribution and takes into account the number of input and output units, or the normalisation implemented in the Kaldi layer (*NaturalGradientAffineComponent* https://github.com/kaldi-asr/kaldi/blob/master/src/nnet3/nnet-simple-component.cc#L2897), sampling from normal distribution and taking into account the number of input units. For the bias vectors $\boldsymbol{b}$ initialisation, we set the parameters to zero, or sample from standard normal distribution.

within a mini-batch. Use back-propagation of error gradients to compute the derivatives with respect to every parameter in every layer, $\theta_i \in \theta$.

- Update every parameter θ_i using accumulated gradients at the point of update u, $g_i(u) = \frac{\partial E}{\partial \theta_i}|_{\theta_i=u}$, and a learning rate $\eta > 0$. The weight change Δ and the updated parameter are given by:

$$\Delta\theta_i(u) = -\eta g_i(u) \tag{2.22}$$

$$\theta_i(u+1) = \theta_i(u) + \Delta\theta_i(u) \tag{2.23}$$

4. Continue until a stopping criterion is met.

In some of the experiments, we use Adam adaptive learning rate method [Kingma and Ba, 2015], to take the history of the weight changes into account and to normalise the learning rate for each weight depending on the magnitude of its gradient. The gradient history is used to choose the update direction, by encouraging the changes in a consistent direction. The gradient magnitude for θ_i reveals if the current region in the error surface is flat or steep at time u, which enables to adapt the learning rate accordingly. Let $\Delta\theta_i(u)$ be the weight change at update time u. The weight change for Adam is defined as:

$$M_i(u) = \alpha M_i(u-1) + (1-\alpha) g_i(u) \tag{2.24}$$

$$S_i(u) = \beta S_i(u-1) + (1-\beta) g_i(u)^2 \tag{2.25}$$

$$\Delta\theta_i(u) = \frac{-\eta}{\sqrt{S_i(u)} + \epsilon} M_i(u) \tag{2.26}$$

where α and β are constants, typically $\alpha = 0.9$ and $\beta = 0.999$.

A stopping criterion in step four can either be a fixed number of *epochs* (completing step three corresponds to an epoch), or one can use *early-stopping*, i.e. stop the training when the error stops decreasing beyond a predefined value. Depending on the experiment, we use either one of the stopping strategies in this thesis.

2.2.2 Regularisation

Training a NN with a large number of parameters and on limited data can lead to overfitting, i.e. the situation when the model fits very well to the training data, but is unable to *generalise* to unseen testing conditions. Multiple *regularisation* approaches have been proposed to control the flexibility of the model and to address this problem.

One way is to add a complexity term to the loss function. L_1 or L_2 penalty term, E_{L_1} or E_{L_2}, can be used as a complexity measure, to penalise larger weights and encourage smoothed representations. In general, p-norm is defined as:

$$E_{L_p} = ||\theta||_p = \big(\sum_i |\theta_i|^p\big)^{\frac{1}{p}} \tag{2.27}$$

L_1 regularisation encourages sparse representations, and L_2 penalises weights proportionally to their magnitude. The loss function becomes:

$$\hat{\mathcal{F}}_{CE}(\theta) = \mathcal{F}_{CE}(\theta) + \gamma E_{\{L_1,L_2\}} \tag{2.28}$$

where γ is a constant and can be optimised on the development set. L_2 regularisation is also referred to as *weight decay*.

Using the ensemble of models can also be used to prevent overfitting. *Dropout* was proposed by Srivastava et al. [2014] to emulate this behaviour with a single model. With dropout, a fraction of units in a NN is dropped at training for each mini-batch. This approach can be more robust than the ensemble of models trained separately, because the sub-networks created by applying dropout share weights. The technique enforces robustness to missing features and it has proved to be effective for many ML tasks, also for ASR.

Normalisation techniques can also have the regularisation effect. *Batch normalisation*, proposed by Ioffe and Szegedy [2015], can be used to normalise the activations, by enforcing zero mean and unit variance over the mini-batch. The statistics (mean μ_b and variance σ_b^2 for example b in the mini-batch of size B), and the normalised representations ($\hat{a}_{ib}$) for each hidden unit are computed as:

$$\mu_b = \frac{1}{B}\sum_{b=1}^{B} a_{ib} \tag{2.29}$$

$$\sigma_b^2 = \frac{1}{B}\sum_{b=1}^{B}(a_{ib} - \mu_b)^2 \tag{2.30}$$

$$\hat{a}_{ib} = \frac{a_{ib} - \mu_b}{\sqrt{\sigma_b^2 + \epsilon}} \tag{2.31}$$

Alternatively, *layer normalisation* proposed by Ba et al. [2016] uses the statistics computed across D features, as:

$$\mu_i = \frac{1}{D} \sum_{i=1}^{D} a_{ib} \tag{2.32}$$

$$\sigma_i^2 = \frac{1}{D} \sum_{i=1}^{D} (a_{ib} - \mu_i)^2 \tag{2.33}$$

$$\hat{a}_{ib} = \frac{a_{ib} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \tag{2.34}$$

Both batch and layer normalisation operations apply a trainable scale and shift parameters, λ_i and ζ_i respectively, on top of normalised activations $\hat{a}_{ib}$. The output after the normalisation transformation for hidden unit i is:

$$z_i = \lambda_i \hat{a}_{ib} + \zeta_i \tag{2.35}$$

Layer normalisation is independent of other training examples, thus can be easily applied to recurrent architectures. Also, the same operation is performed at training and at test time. For batch normalisation, the statistics computed over the whole training set are typically used at test time.

Another technique which can be seen as regularisation is *data augmentation*. In general, generating additional noisy examples enables the model to generalise better by avoiding overfitting to the original training set. For speech recognition, *multi-condition training* is a form of data augmentation. Generating multiple additional acoustic conditions for training set augmentation proved to be an effective approach. In Kaldi [Povey et al., 2011], the training time-domain utterances can be augmented by reverberation, speed and volume perturbation, or adding different background noises at different SNRs. Signal-to-Noise Ratio (SNR) is defined as the ratio of the power of the signal to the power of the noise, and is usually expressed in decibels (dB).

Recently, an augmentation method in the spectral domain, SpecAugment, have been proposed for ASR by Park et al. [2019]; Park et al. [2020]. This approach consists in masking time and frequency bins of the input FBANK representations, hence it resembles some of the data augmentation techniques used

by CNNs for computer vision tasks, e.g. random patch cropping as used by Krizhevsky et al. [2012a].

2.2.3 Supervision

The alignment of a word sequence $\boldsymbol{w}$ with the acoustics provides labels $\boldsymbol{y}$, which are used to train hybrid HMM-NN models. In *supervised* training, the transcripts are provided and are accurately corresponding to the acoustics. This can be achieved by *supervised speaker enrolment*, when the speakers read text which is directly used as supervision [Liao, 2013], however, this approach requires relatively large amount of speaker-specific data to cover the acoustic space and is costly and time-consuming. Another solution for obtaining accurate transcripts is *manual audio transcription*, which does not require additional recordings, but is also expensive, especially with professional annotation [Novotney and Callison-Burch, 2010]. An example of a speech recognition benchmark corpus for supervised training is WSJ (Wall Street Journal) [Paul and Baker, 1992] – a corpus of read speech. In this scenario, the transcripts are accurate and are available for training.

Nevertheless, to obtain a robust ASR system for a new domain, acoustic model retraining is often necessary to better match the deployment conditions. Since NN acoustic models require large volumes of training data, a surge in partially supervised AM research has been observed in recent years. In this thesis we focus on supervised training[7]. However, for a more complete introduction to deep learning for acoustic modelling, we briefly summarise current alternatives to supervised acoustic model training, and we provide the references to the papers where those approaches are evaluated for ASR.

- *Active learning* in the context of ASR consists in automatic selection of the data to be manually transcribed. Different data selection criteria were previously studied by Hakkani-Tür et al. [2002]; Riccardi and Hakkani-Tür [2005]; Yu et al. [2009]; Itoh et al. [2012]; Drugman et al. [2016]; Long et al. [2018].

- In *lightly-supervised* training the transcripts are available, but are not well matched to the acoustics. In this scenario, the challenge is to select match-

[7]Here, supervision relates to the training set transcription, rather than the transcription for the adaptation or test set, or the information about the underlying factors of variation.

ing excerpts of text and acoustics [Lamel et al., 2002; Bell and Renals, 2015a; Veselý et al., 2018]. Alignment with imperfect transcripts has an application e.g. for broadcast media to align the acoustics with the captions.

- *Semi-supervised* training is a process of training with automatic transcriptions produced by a speech recognition system and was explored e.g. by [rahman Mohamed et al., 2009; Veselý et al., 2013; Thomas et al., 2013; Zhang et al., 2014; Manohar et al., 2015; Baskar et al., 2019; Karita et al., 2019; Krishnan Parthasarathi and Strom, 2019]. Automatic transcription is relatively cheap, but not always reliable. The challenge for semi-supervised training is therefore to identify the most reliably transcribed data to be used for training.

- *Weakly-supervised* methods try to make the use of labels which are not well matched and may only be loosely related to the acoustics, as surrogate for ground truth labels. For example, Singh et al. [2020] use solely video title and post text as additional contextual information for acoustic model training.

- We refer to the methods as *softly-supervised* when the soft labels for training are generated with another model. Teacher-student approach [Hinton et al., 2015; Li et al., 2014; Wong and Gales, 2016; Li et al., 2017; Watanabe et al., 2017; Manohar et al., 2018] and self-teaching networks [Lu et al., 2019] fall into this category. In teacher-student training, a student model is trained to reproduce the outputs of a teacher model. In self-teaching networks, soft labels are used within the same network – lower layer is trained to mimic the behaviour of the output layer.

- *Self-supervision* is defined as autonomous supervised learning, i.e. the targets for supervised training are computed from the input signal [Pascual et al., 2019; Kurata and Audhkhasi, 2019; Stafylakis et al., 2019; Schneider et al., 2019; Ravanelli et al., 2020]. Self-supervised training leverages large amounts of unlabelled data, while learning representations in a supervised manner.

- *Unsupervised* learning aims to extract meaningful latent representations of speech without any labels [Park and Glass, 2008; Jansen et al., 2010; Kam-

per et al., 2017; Chung and Glass, 2018; Hermann et al., 2018; Chorowski et al., 2019]. In the literature, those approaches are also referred to as *zero-resource* or *acoustic unit discovery* [Dunbar et al., 2019].

2.2.4 Architectures for acoustic modelling

In this thesis, we refer to a family of multi-layer Neural Networks as NN. Two main types of the architectures can be differentiated: feed-forward and recurrent neural networks. Feed-forward models do not have feedback connections to feed back the information, whereas recurrent neural networks include those connections. The focus of this thesis are feed-forward architectures with different connectivity patterns (Figure 2.3). In this section, we give a brief overview of deep architectures most commonly used for phonetic context modelling in ASR.

Deep Neural Networks

In this thesis, we use the term Deep Neural Network (DNN) to refer to a feed-forward fully-connected structure, i.e. a model where each layer has a full connectivity with the previous layer. Let $\boldsymbol{x} = (x_1, x_2, ..., x_D)$ be the input feature vector. Each activation vector value of a fully-connected layer can be computed as:

$$a_j = \sum_{i=1}^{D} w_{j,i} x_i + b_j \tag{2.36}$$

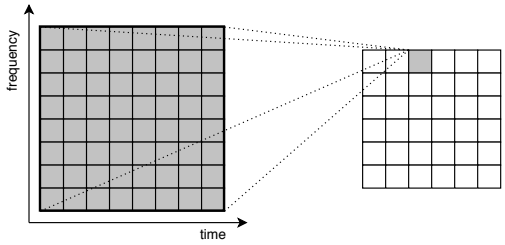
where $w_{j,i}$ is an element of the weight matrix $\boldsymbol{W}$, i.e. the weight for the input feature i and the output unit j, and b_j is an element of the bias vector $\boldsymbol{b}$.

DNNs have been widely used for modelling the acoustic context [Hinton et al., 2012; Maas et al., 2017]. In such models, the input vector is a concatenation of the feature vectors for multiple neighbouring frames within context c:

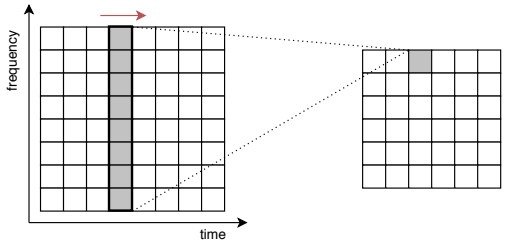
$$\hat{\boldsymbol{X}}_{\boldsymbol{t}} = (\boldsymbol{x}_{t-c}, ..., \boldsymbol{x}_t, ..., \boldsymbol{x}_{t+c}) \tag{2.37}$$

Time-Delay Neural Networks

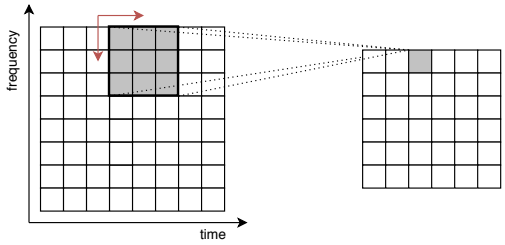
Time-Delay Neural Network (TDNN) was proposed to better model long temporal context [Waibel et al., 1989; Lang et al., 1990; Peddinti et al., 2015b]. This architecture introduces a local connectivity pattern with the parameters tied across time, hence it can be seen as a Convolutional Neural Network (CNN) with 1D



(a) DNN



(b) TDNN



(c) CNN

Figure 2.3: Connectivity patterns for different feed-forward architectures. The movement of the kernel is denoted with a red arrow. We assume a single 8×8 input feature map, and a single output channel for the CNN. The weights are shared across time for a TDNN, and across time and frequency for a CNN.

kernels[8]. TDNNs apply convolutions in time in a hierarchical manner and thus are able to exploit variable-length contextual information. The effective receptive field of the upper layers of a TDNN model is larger than a receptive field of the lower layers. Moreover, the window of TDNN layers activations is sub-sampled, which reduces the number of parameters, and thus can be seen as a *dilated* 1D convolutional operation.

[8]Nevertheless, in this thesis we use the term CNN to refer to the models with 2D kernels, described later in this section.

Convolutional Neural Networks

CNNs take account of the local input structure by using *local receptive fields* to model spatially local correlations. Combining this local modelling with *weight sharing* and *pooling* enables invariances to be exploited across the structure of the input, typically leading to better generalisation to unseen data compared to fully-connected DNNs.

CNNs have been applied to speech recognition by treating time-frequency representations analogously to images [Sainath et al., 2013b; Abdel-Hamid et al., 2014a; Sainath et al., 2013a; Swietojanski et al., 2014; Huang et al., 2015; Sainath et al., 2015b]. The feature map input to CNN acoustic models typically employs a sub-sequence of FBANK features (often concatenated with their first and second temporal derivatives) arranged in a two-dimension array (*feature map*) whose size is $time \times freq$ (where $time$ is the width of the time context window and $freq$ is the number of frequency bins in the FBANK).

CNN exploits local time-frequency relationships in the data by using local receptive fields. For ASR, localised convolutions enable to model invariances to slight shifts in time and frequency [Maas et al., 2017]. Convolutions across frequency can perhaps improve the robustness to different speakers and speaking styles which may be manifested as variations in activity in different frequency bands. Convolutions across time can be beneficial in reverberant environments, where temporal artifacts are introduced or to account for speaking rate variation [Zhao et al., 2015; Mitra and Franco, 2015].

Sharing the weights across a feature map amounts to extracting the same feature at all points of the input feature map. For ASR, this may enable the network to model the situation in which noise or distortion is more apparent in some bands of the spectrum than in others, allowing representations to be computed from the cleaner parts of the spectrum.

Moreover, weight sharing and the use of local receptive fields reduces the number of the parameters of the model which can help with overfitting.

Each feature map activation value $a_{i,j}$ in a CNN model is computed as

$$a_{i,j} = \sum_{k=0}^{m-1} \sum_{l=0}^{m-1} w_{k,l} x_{i+k,j+l} + b_{i,j} \tag{2.38}$$

$$= f(\boldsymbol{W} \otimes \boldsymbol{X} + b_{i,j}), \tag{2.39}$$

where $w_{k,l}$ are elements of the shared $m \times m$ weight matrix $\boldsymbol{W}$, $b_{i,j}$ is the shared bias, and $x_{i+k,j+l}$ is the input at position $i+k, j+l$. We use k and l to index into the receptive field, whose top left corner is at $x_{i,j}$. $\boldsymbol{X}$ denotes a matrix of input values within local receptive field. $\otimes$ denotes cross-correlation and this is the operation performed by convolutional layers in our models.[9]

Padding input representations with zeros around the border enables the size of the output feature map to match the size of the input feature map, which is important for deep convolutional architectures; moreover it ensures that information is not lost from the edges of the input feature map. This is sometimes referred to as SAME padding, in contrast to a VALID convolution which does not use padding and thus results in a smaller output feature map.

Shallower CNN acoustic models have tended to use valid convolution [Sainath et al., 2013b; Abdel-Hamid et al., 2014a; Swietojanski et al., 2014], however more recently investigated models [Sercu and Goel, 2016a; Yu et al., 2016a; Qian and Woodland, 2017] have used SAME padding to preserve the feature map size.

CNN classifiers require some form of compression or down-sampling to map a feature map to a classification. This is most often achieved through the use of *pooling* layers between convolutional layers. Pooling layers discard the exact positional information of a feature, and may be regarded as smoothing filters. In practice two types of pooling operations are used in CNN models: max and average pooling.

Max pooling is sensitive to the existence of some feature in sub-region of the initial representation, and average pooling measures the mean value of existence of a feature in that sub-region. In practice, max pooling has shown better empirical results than average pooling in many pattern recognition tasks.

Alternatively, a feature map can be down-sampled using a convolutional layer with a larger *stride*. For image recognition, it has been demonstrated that replacing a max-pooling layer by a convolutional layer with increased stride does not reduce the classification accuracy [Springenberg et al., 2015]. Using convolutional layers instead of pooling layers to down-sample feature maps can also be seen as learning the pooling operation rather than fixing it.

[9] If the cross-correlation kernel $\boldsymbol{w}$ is flipped horizontally and vertically, then (2.39) becomes a convolution. The network learns the kernel appropriate to its orientation – so if convolution is implemented with a flipped kernel, it will learn that it is a flipped representation. The specific properties of convolution but not of cross-correlation (commutativity and associativity) are not required for a CNN acoustic model.

Both these approaches to down-sampling maintain translation invariance and offer a degree of smoothing, and in a speech recognition context can offer robustness by compensating for variation in the frequency domain arising from the acoustic environment or speaker characteristics. In addition down-sampling can also reduce the complexity of the model.

Convolutional Neural Networks were the one of the first successfully used deep neural network architectures, originally used for image processing, computer vision, and document understanding [LeCun et al., 1989a, 1998a], and since about 2012 they have defined the state-of-the-art for many computer vision tasks [Krizhevsky et al., 2012a; Sermanet et al., 2013]. Very deep convolutional neural networks (deep CNNs) have been shown to improve the recognition accuracy compared to CNNs with fewer layers for both image recognition [Simonyan and Zisserman, 2015; Szegedy et al., 2015] and speech recognition [Sercu et al., 2016a; Sercu and Goel, 2016a; Yu et al., 2016a; Qian and Woodland, 2017]. The key concept of deep CNNs is to replace a single kernel from a classical CNN model with a stack of convolutional kernels of smaller size. Such a stack of small kernels can have a similar effective receptive field as a single larger kernel but with a reduced number of independent parameters. Moreover, stacking convolutional layers enables more complex features to be learned due to the additional non-linearities in the model.

There is experimental evidence that convolutional structure of CNNs allows better solutions to be learned compared with DNNs. For instance, Huang et al. [2015] estimated both CNN and DNN acoustic models on a speech recognition task, training both networks on about 1 000 h of data – an amount possibly big enough for a DNN to learn all the necessary invariances. The CNN resulted in reduced word error rates compared to the DNN; moreover for a distant speech recognition task the gain of CNNs over DNNs increased in direct proportion to the speaker-microphone distance.

Earlier works on CNNs for speech recognition applied convolutional filters solely across the frequency axis either sharing the weights for all frequency bands or with limited weight sharing using separate sets of weights for different frequency bands [Abdel-Hamid et al., 2013, 2014b]. More recent works have shown good performance of CNN models which convolve in both time and frequency [Tòth, 2014], including very deep networks using stacked small convolutional filters [Sercu et al., 2016a; Sercu and Goel, 2016a; Yu et al., 2016b; Xiong et al.,

2017; Tan et al., 2018].

Deep CNN models with small two-dimensional kernels, designed for image recognition [Simonyan and Zisserman, 2015; Krizhevsky et al., 2012a; Szegedy et al., 2015], have recently been investigated for various speech processing tasks.

Empirical results have shown that the performance of deep CNNs is comparable to LSTM Recurrent Neural Networks (RNNs) [Xiong et al., 2017] and compatible to bidirectional LSTM RNNs [Xiong et al., 2016]. The feed-forward nature of CNN models results in lower latency and therefore may be preferable in real-time scenarios [Yu and Li, 2017].

The use of deep CNNs for noise-robust speech recognition was investigated by Qian and Woodland [2017] for Aurora-4 robust speech recognition and AMI distant speech recognition corpora, with a specific focus on optimal deep CNN architectures – the kernel sizes, pooling and padding strategies, and the size of the input feature map. Tan et al. [2018] propose very deep convolutional residual network for robust ASR. They also explore adaptation and adaptive training for this deep CNN architecture.

Recurrent Neural Networks

The Recurrent Neural Network (RNN) was designed to model sequential data, such as speech. This architecture is widely used for acoustic modelling [Robinson et al., 1996]. Although we do not use recurrent architectures in this thesis, we briefly describe this model for completeness.

In a model using recurrent connections, hidden units at time t are connected with the input feature vector at time t, as well as with its hidden state at time $t-1$. Hidden units in an RNN act as memory units. They also compress the information, providing a way to model long term dependencies. A bi-directional version of a recurrent architecture connects two hidden layers of the opposite directions with the same output. A bi-directional model has therefore the access to the past, as well as future samples within an utterance.

For ASR, Long Short-Term Memory (LSTM) models are typically used, to avoid the *vanishing gradient* problem, i.e. the situation when the gradients are very small and the training is hindered. LSTM uses *gates* to control the flow of information, depending on the current input and the previous state. We refer the reader to [Graves et al., 2013] for the details about a bi-directional LSTM acoustic model.

2.2.5 End-to-end systems and Transformers

Recently, the models which directly map from the sequence of observations $\boldsymbol{X}$ to the sequence of words $\boldsymbol{w}$, referred to as "end-to-end", are gaining a lot of attention. In a sense, HMM models trained with ML can be regarded as end-to-end – they estimate $P(\boldsymbol{X}|\boldsymbol{w})$ and when combined with the language model, they give an estimate of $P(\boldsymbol{w}|\boldsymbol{X})$ [Renals, 2019]. Also, MMI sequence disriminative training can be seen as end-to-end – the objective is utterance rather than frame level. Lattice-free MMI [Povey et al., 2016] was proposed to directly[10] and efficiently[11] compute HMM state posteriors with a single TDNN acoustic model. Combination with the language model gives an estimate of $P(\boldsymbol{w}|\boldsymbol{X})$.

Sometimes, a model is considered as "end-to-end" when it is fully-differentiable and models every component of speech recognition (acoustics, language, lexicon) with a single neural network. In this sense, the recently proposed approaches to directly map input to output sequences, are: CTC [Hannun et al., 2014], Atttention-based Encoder-Decoder [Chan et al., 2016], RNN Transducer [Graves, 2012], and Transformer [Vaswani et al., 2017]. The first three approaches are based on recurrent neural networks, while Transformers use self-attention mechanism. An empirical comparison of end-to-end models used for large scale ASR can be found in [Li et al., 2020b]. In this section, we briefly describe the mechanisms used by a Transformer, and we review Transformer-based ASR systems, focusing on their connections to CNNs.

Transformers are based on a multi-head self-attention mechanism. Attention was introduced in Encoder-Decoder architectures. The role of the Encoder is to generate high level representations, while the Decoder generates output labels. Attention network connects the acoustic Encoder with the Decoder. It acts on the whole Encoder output, enhancing the Decoder with a weighted sum of the Encoder hidden states. Attention-based models were initially proposed for Machine Translation (MT) [Bahdanau et al., 2015], and later adapted for ASR [Chorowski et al., 2015; Chan et al., 2016].

Transformer architectures use self-attention as a building block for the Encoder and the Decoder. The Encoder blocks consist of Multi-Head Self-Attention (MHSA) and feed-forward layers. The Decoder blocks employ an additional Encoder-Decoder Attention layer. Figure 2.4 shows an example of a Transformer

[10] Without frame-based CE pre-training.
[11] Without pre-computing lattices for the denominator.

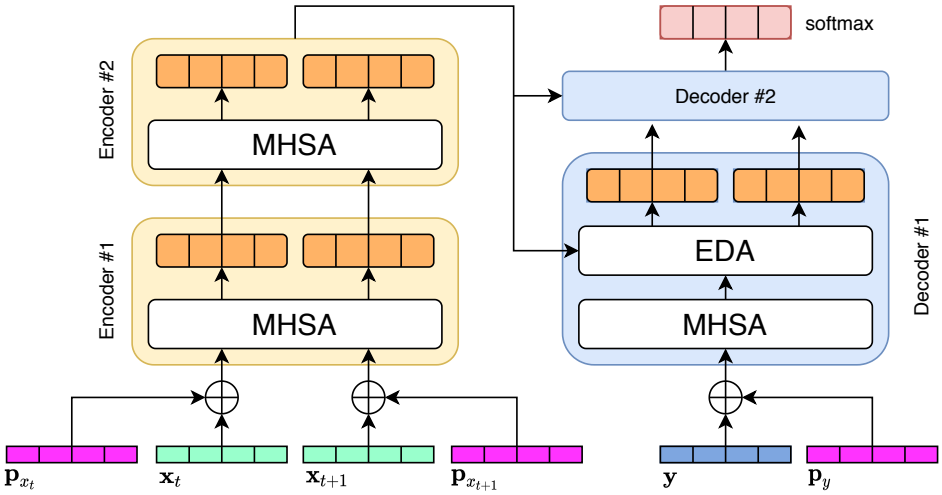


Figure 2.4: An example of a Transformer model with two Encoders, two Decoders, and two input feature vectors ($\boldsymbol{x}_t$ and $\boldsymbol{x}_{t+1}$). Positional encoding vectors are denoted with $\boldsymbol{p}$ and the Decoder previous output encoding is denoted with $\boldsymbol{y}$. *EDA* stands for Encoder-Decoder Attention and *MHSA* for Multi-Head Self-Attention.

model with two stacked Encoders and Decoders and two input feature vectors. We omit batch or layer normalisation, dropout, and residual connections for simplicity.

Self-attention mechanism provides a way to learn dependencies between arbitrary positions in the input space. It is computed between every pair of frames in the input sequence, enabling to associate a current frame t with every other frame τ in an utterance. When used in the Encoder, self-attention allows to look at other positions in the input sequence while generating high level representation for the current frame. Let $\boldsymbol{x}_t, \boldsymbol{x}_\tau \in \mathbb{R}^{d_i}$ be input feature vectors and $\boldsymbol{z}_t \in \mathbb{R}^{d_o}$ be the output embedding vector at time t. The output for frame t in an utterance of length T is defined as:

$$\boldsymbol{z}_t = \texttt{self-attention}(\boldsymbol{x}_t) = \sum_{\tau=1}^{T} \alpha_{t\tau} \cdot \boldsymbol{x}_\tau \boldsymbol{W}_v. \tag{2.40}$$

The attention probability $\alpha_{t\tau}$ is a `softmax` function of the scaled attention score $a_{t\tau}$:

$$\alpha_{t\tau} = \texttt{softmax}(\beta a_{t\tau}) = \frac{\exp(\beta \cdot \boldsymbol{x}_t \boldsymbol{W}_q \boldsymbol{W}_k^T \boldsymbol{x}_\tau^T)}{\sum_{\tau'} \exp(\beta \cdot \boldsymbol{x}_t \boldsymbol{W}_q \boldsymbol{W}_k^T \boldsymbol{x}_{\tau'}^T)} \tag{2.41}$$

Self-attention layer is therefore parameterised by three transformation matrices: $\boldsymbol{W}_q$, $\boldsymbol{W}_k$, $\boldsymbol{W}_v$. $\boldsymbol{W}_q$ and $\boldsymbol{W}_k \in \mathbb{R}^{d_i \times d_k}$ transform $\boldsymbol{x}_t$ to *query* and *key* space, and

$\boldsymbol{W}_v \in \mathbb{R}^{d_i \times d_o}$ transforms input vectors to *value* space. $\beta = \frac{1}{\sqrt{d_i}}$ is a scaling factor.

By summing over the whole input sequence in Equation 2.40, $\boldsymbol{z}_t$ becomes invariant to the input order permutations. To incorporate the knowledge about the order of the frames in the sequence, various *positional encodings* have been proposed in the literature. For example, Dong et al. [2018] use sinusoidal positional encodings, as proposed in the original Transformer paper [Vaswani et al., 2017], and Povey et al. [2018] use one-hot relative positional encodings[12].

In general, if the attention score is defined as:

$$a_{t\tau} = \boldsymbol{x}_t \boldsymbol{W}_q \boldsymbol{W}_k^T \boldsymbol{x}_\tau^T, \tag{2.42}$$

it can be redefined to include the positional information as:

$$a_{t\tau} = (\boldsymbol{x}_t + \boldsymbol{p}_t) \boldsymbol{W}_q \boldsymbol{W}_k^T (\boldsymbol{x}_\tau + \boldsymbol{p}_\tau)^T, \tag{2.43}$$

where $\boldsymbol{p}_t, \boldsymbol{p}_\tau \in \mathbb{R}^{d_i}$ are the positional encoding vectors. They can be any vector representations of the position of the input feature vector $\boldsymbol{x}_t$.

Transformers use Multi-Head Self-Attention (MHSA) to enables the model to focus on different positions, with different representation subspaces. H self-attention heads are used to learn alternative representations from different parts of the input, by using different query, key and value matrices. The output embedding vector $\boldsymbol{e}_t$ is a concatenation of vectors $\boldsymbol{z}_t$ for each head h, transformed to a common space with an affine function as:

$$\boldsymbol{e}_t = \texttt{MHSA}(\boldsymbol{x}_t) = \left[\boldsymbol{z}_t^{(h)} \right]_{h \in H} \boldsymbol{W}_o + \boldsymbol{b}_o \tag{2.44}$$

where $\boldsymbol{W}_o \in \mathbb{R}^{H d_h \times d_o}$, $\boldsymbol{b}_o \in \mathbb{R}^{d_o}$.

MHSA layer is similar to a convolutional layer. Intuitively, every attention head can learn to attend to a given relative shift around each point in the input space. The value matrix $\boldsymbol{W}_v^{(h)}$ is therefore analogous to the convolutional kernel for each shift, and the relative positional encoding introduces dependence of the attention score on the shift, i.e. it enables the equivariance to translation property.

Cordonnier et al. [2020] showed that a self-attention layer is a generalisation of a convolutional layer, given enough heads and using relative positional encoding. The authors provide a theoretical proof that *"a multi-head self-attention layer*

[12] Alternatively, convolutional layers can be used at the Transformer front-end to encode relative frame positions, as e.g. in [Yeh et al., 2019; rahman Mohamed et al., 2019].

with H heads of dimension d_h, output dimension d_o and a relative positional encoding of dimension $d_p \geq 3$ can express any convolutional layer of kernel size $\sqrt{H} \times \sqrt{H}$ and $\min(d_h, d_o)$ output channels.". They also show that in practice (for vision tasks) self-attention layers learn convolutional filters.

The main difference between self-attention and CNN layers is in their receptive field. In a self-attention layer, the receptive field is typically the whole input space. For a CNN, the region of the input space that affects a particular hidden unit is restricted by the size of the kernel. However, the effective receptive field of CNNs can be enlarged by using very deep networks with stacked CNN layers, and with the use of dilation.

The Transformer model was first proposed for Natural Language Understanding (NLU) [Vaswani et al., 2017], and achieved state-of-the-art results in those sequence-to-sequence tasks. Encouraged by those results, and by computational benefits (more parallelisation is possible than for recurrent architectures), Transformer was also evaluated in other domains, also for ASR. Transformer-based ASR architectures were used in ASR for a single-model end-to-end modelling (as e.g. by Pham et al. [2019]), and in a hybrid setup [Wang et al., 2020].

For speech recognition, the input sequence is much longer than a sequence of words for NLU. Due to quadratic computational complexity of the self-attention layer, as well as correlation between acoustic feature vectors, Povey et al. [2018] proposed to stack frames within a fixed size window for self-attention computation. Another approach to reduce the input sequence length is down-sampling. Convolutional layers have been previously used for this purpose, e.g. by Dong et al. [2018]; Tsunoo et al. [2019].

Convolutional layers can also be used in Transformer-based architectures to encode the position of the frames in the input utterance. Yeh et al. [2019] and rahman Mohamed et al. [2019] model the context with 2D causal convolutions, to implicitly encode relative positions and learn short-range time-frequency patterns. Wang et al. [2020] argue that convolutional embeddings implicitly perform frame stacking, which can be used for down-sampling, and to encode relative positional information.

Similarly, Wang et al. [2020] use 2D convolutions for positional encoding in a hybrid Transformer model. The authors compare convolutional encodings with the original sinusoidal encodings, as well as frame stacking, and show superior results for convolutional layers in a hybrid ASR setup. In this work, an iterated

loss (i.e. an interpolation of auxiliary CE losses at intermediate layers with the original CE loss) was also crucial to train deep transformer-based AMs. To enable streaming ASR, the authors also use limited right context R at inference, i.e. $\tau > \tau + R$ is masked to $-\infty$. Sufficient depth of Transformer-based models is another important factor for optimal performance, as showed by Wang et al. [2020] for a hybrid model, and by Pham et al. [2019] for an Encoder-Decoder end-to-end model.

In summary, the learning mechanism of a multi-head self-attention is similar to convolution. Moreover, both layer types can be used in conjunction within a single acoustic model, to combine the advantages of both of them. 2D convolutional layers can be used for down-sampling and positional encoding, to facilitate optimisation, while a multi-head self-attention is a powerful, efficient and parallelisable method to model long-term temporal relationships between input feature representations, providing state-of-the-art results.

2.3 Summary

In this chapter, we reviewed ASR methods, with the focus on hybrid HMM-NN models. We first outlined the hybrid approach and we described the compound components and processing stages. We then briefly reviewed NNs and their integration into ASR. We described the training procedure, regularisation, supervision, as well as different architectures used for acoustic modelling.

CHAPTER 3

Speech corpora

Throughout the thesis we use five different corpora for our experiments. All of them are in English. By testing our methods on multiple datasets, we aim to address different underlying factors of variation, and to improve the generalisation of our findings. Aurora-4 was chosen to evaluate the performance of our methods for speech with different types of additive noises, recorded with different microphones. AMI corpus enabled to test the approaches in more realistic multi-speaker meeting scenarios, where room acoustics, different native languages of the speakers, and different microphones contribute to signal variability. MGB and Financial and Political News datasets were used to test our methods in more challenging broadcast conditions (with unreliable labels for some of the training data, lack of speaker or background noise annotation). We also use VoxCeleb corpus to evaluate proposed utterance embeddings for speaker verification, on a dataset well established for this task.

3.1 Aurora-4

Aurora-4 [Parihar and Picone, 2002] is a medium-vocabulary task based on the Wall Street Journal (WSJ) corpus [Paul and Baker, 1992] – a corpus of read sentences from the Wall Street Journal, recorded under clean conditions. Aurora-4 has different noise types artificially added, to study the effects of variation in

microphone and noise for speech recognition.

Aurora-4 noise types are:
- car
- crowd of people (babble)
- restaurant
- street
- airport
- train station

The corpus contains two parallel training sets – clean and multi-condition (Figure 3.1). Each training set contains 7 138 utterances (around 15 hours) from 83 speakers. The clean-condition training set is equivalent to the SI-84 subset of WSJ corpus, with an overall audio duration of about 15h. The multi-condition training set is corrupted with noise and recorded with a different microphone. Its overall duration is equal to the clean-condition training set duration. It consists of selected utterances from the clean-condition training set (recorded with the primary Sennheiser microphone) and utterances from 18 different microphone types (with the same linguistic content). 75% of utterances recorded with a primary microphone are corrupted using six noise types (car, babble, restaurant, street, airport, train) at different SNR levels (10-20 dB). The same noise distortion is applied to the mismatched microphone subset. Unless otherwise stated, in our experiments we use the GMM-HMM forced alignments generated from multi-condition training set.

The test set[1] is formed from 4 subsets: 330 clean utterances selected from SI-84 WSJ corpus, 330 × 6 utterances with 6 types of additive noise at 5-15 dB SNR, 330 utterances recorded with a different microphone, 330 × 6 utterances recorded with a different microphone and with 6 types of additive noise at 5-15 dB SNR. Those test set subsets are referred to as A, B, C, and D, respectively. The overall test set audio duration for all four subsets is about 9 hours.

We also evaluate Aurora-4 for the task of speaker, acoustic condition, noise, and gender verification. The data split for the verification tasks is depicted in Figure 3.2. We take the complete multi-condition test set (subsets A, B, C, D) and randomly select the utterances for the enrolment (enrol) and evaluation (eval) subsets. We ascertain that all acoustic conditions and all speakers are represented in both subsets. In Aurora-4 experiments we use the task-standard bi-gram language model.

[1] We use test_eval92 from the default Kaldi split (kaldi/egs/aurora4/s5/local/aurora4_data_prep.sh) for evaluation.

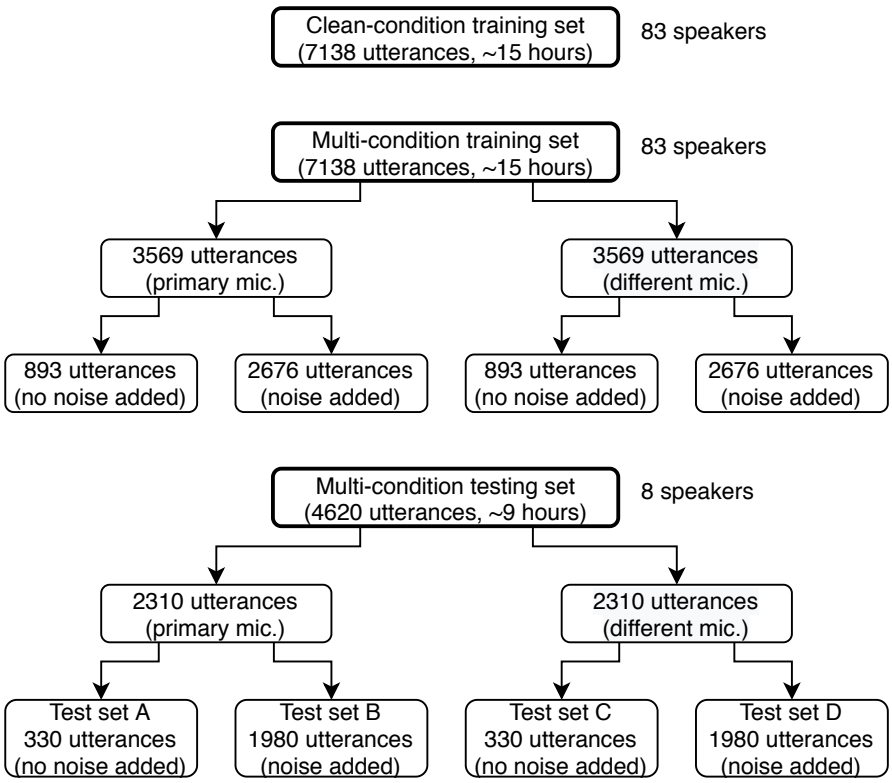


Figure 3.1: Aurora-4 corpus speech recognition split.

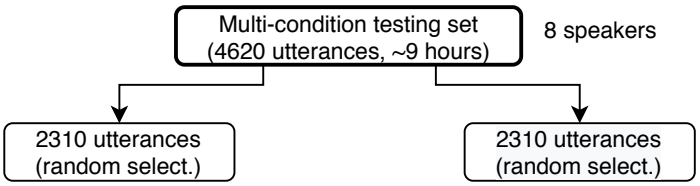


Figure 3.2: Aurora-4 corpus verification split. *Random select.* indicates random selection of the utterances.

3.2 AMI

The AMI meeting corpus [Renals et al., 2007] contains around 100 hours of meetings recorded at three sites in Europe (Edinburgh, IDIAP, TNO). The meetings are scenario-based (around 70% of the corpus) or spontaneous non-scenario-based. The speakers are non-native in large proportion. The native language is English for 42% of speakers across the whole corpus, Dutch for 27%, and other language for 31%. The speech is captured by an Individual Headset Microphone (IHM) and an 8-microphone array, referred to as Multiple Distant Microphones (MDM). The microphones are combined using the BeamformIt [Anguera et al., 2007] toolkit. For a Single Distant Microphone (SDM) scenario, only the first microphone of

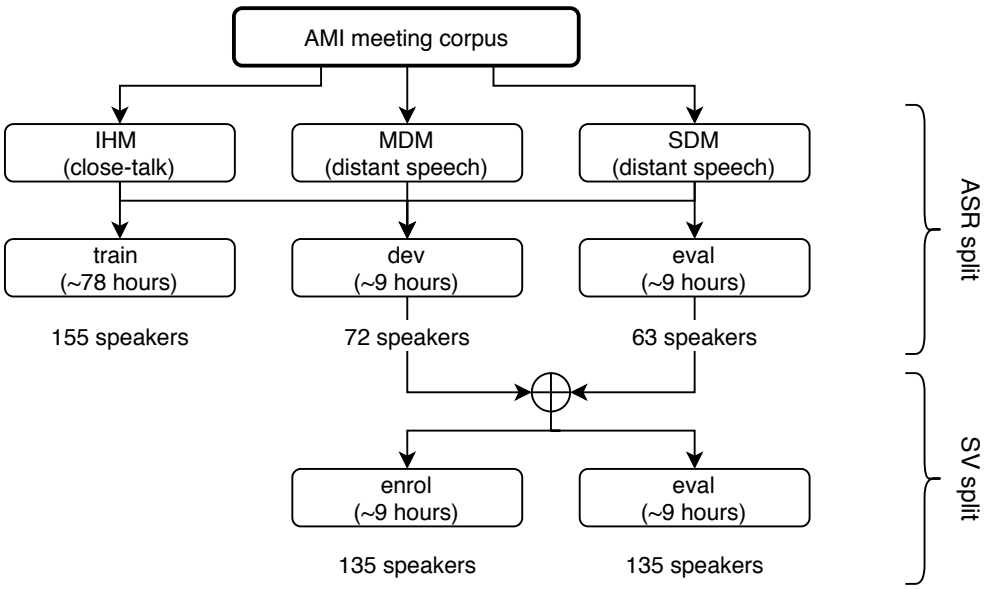


Figure 3.3: AMI meeting corpus splits. *ASR split* indicates the speech recognition data split, and *SV split* the speaker verification split. IHM stands for Individual Headset Microphone, SDM for Single Distant Microphone, and MDM for Multiple Distant Microphones.

the array is used.

We use the suggested train/dev/eval data split [Swietojanski et al., 2013] (Figure 3.3), we train on the training set partition, and we evaluate the models on both dev and eval sets. The speakers do not overlap in the partitions. For decoding we use the 3-gram language model from the standard recipe[2], which is an interpolation of the models trained on the AMI and Fisher English transcripts.

We use AMI IHM for speaker verification in some of our experiments. We merge dev and eval sets from the original ASR split and we prepare enrolment (enrol) and evaluation (eval) subsets, by randomly selecting utterances. We only use close-talk recordings in speaker verification experiments.

3.3 MGB

We evaluate some of the proposed approaches on the Multi-Genre Broadcast (MGB) datasets from the MGB Challenge competitions[3] in 2015 (MGB-1) [Bell et al., 2015] and 2017 (MGB-3). We only use English recordings.

[2] kaldi/egs/ami/s5b/run.sh
[3] http://www.mgb-challenge.org/

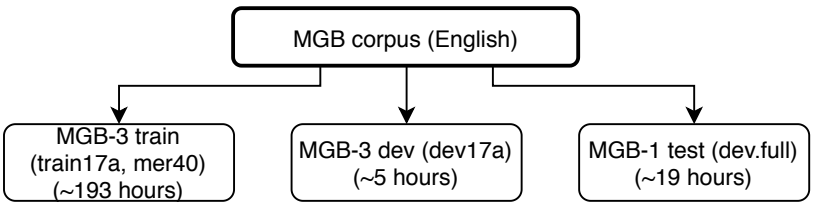


Figure 3.4: MGB corpus speech recognition split.

MGB contains TV recordings of the whole range of BBC programmes, e.g. documentary, drama, news, comedy, competition. It is a very challenging dataset due to imperfect transcriptions for the training set, overlapping speech, and various types of background noise.

The partitions used in our experiments are depicted in Figure 3.4. We train our models on the MGB-3 training set (train17a), which contains recordings from 540 TV episodes in English. To select the training data we use Word Matching Error Rate (WMER) as a measure indicating the quality of the labels. First, lightly supervised alignment is performed to align training transcripts, as described by Bell et al. [2015]. WMER is obtained by scoring the decoding against the aligned transcripts. We only use the segments with WMER $\leq 40\%$ for training, resulting in around 193 hours of data.

We evaluate our models on both the MGB-3 development set (dev17a) and MGB-1 test set (dev.full). The MGB-3 dev set consists of about 5 hours and MGB-1 test set of about 19 hours of multi-genre TV episodes.

There is no speaker annotation for the MGB dataset. In the metadata, there is however information about the colour of the caption (one of four colours), corresponding to the colours manually assigned by one the subtitlers while preparing the captions for broadcasting. We use a 3-gram language model in the MGB experiments, with lattices being later re-scored with a 4-gram language model.

3.4 Financial and Political News

We also use a larger proprietary collection of broadcast TV financial and political news domain corpus for evaluation (around 830 hours of training data). We make the use of speaker clustering information provided. The speakers in the test set are clustered, and also manually annotated.

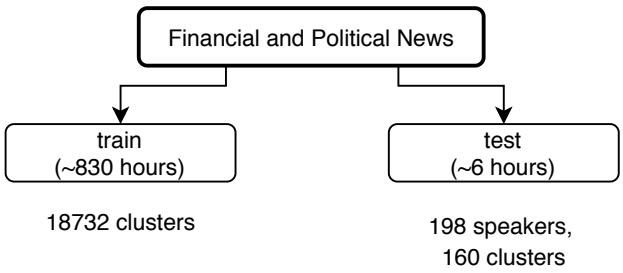


Figure 3.5: Financial and Political News corpus speech recognition split.

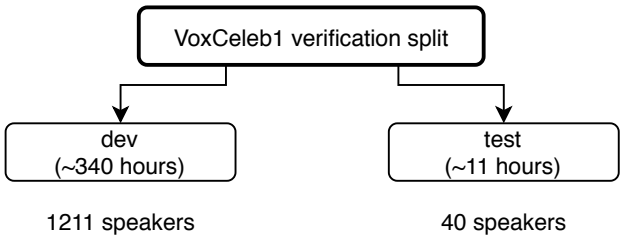


Figure 3.6: VoxCeleb1 corpus speaker verification split.

3.5 VoxCeleb1

VoxCeleb[4] [Nagrani et al., 2017] is an audio-visual corpus consisting of short clips, extracted from YouTube interview videos. VoxCeleb has two parts. We only use the first part (VoxCeleb1) in our experiments. VoxCeleb1 contains over 100 000 utterances for 1 251 celebrities (speakers). We use the recommended speaker verification data split (Figure 3.6).

[4] http://www.robots.ox.ac.uk/~vgg/data/voxceleb/

CHAPTER 4

Embeddings for attribute-aware and adaptive training

4.1 Overview

This chapter is based on [Rownicka et al., 2017], [Rownicka et al., 2018] and [Rownicka et al., 2019] and proposes different embeddings and their transformations for acoustic model adaptation, to improve the robustness of ASR systems.

Using embeddings is a way to provide the acoustic model with the information about the underlying factors of variation, whose source is different to frame-level differences between spoken sounds. Providing the model with this complementary information, and therefore separating latent causes of variation in the data, could help to disentangle the overall representations and lead to more robust learning. In this chapter, we explore attribute-aware, as well as adaptive training, to provide more flexibility in selecting the most informative features. We optimise the parameters of the models to account for high variability of speech. The embeddings are utterance or frame-wise hence the models can quickly adapt to testing conditions in an unsupervised manner, without the need for a separate adaptation dataset.

We explore the following research questions:

- How does the type of the embedding influence the ability to learn robust representations?
- Can the incorporation of the embedding be optimised?
- Is the normalisation of hidden representations more effective than normalisation of the input feature space?

In this chapter, we first give a literature overview of the related work (Section 4.2). We then describe the proposed methods in Section 4.3. In Section 4.4 we describe the experimental setup and we provide the results. Finally, we discuss and conclude the results in Section 4.5.

4.2 Background

Neural Network models for ASR construct probability distributions by learning from input data. Therefore, the quality of the input data, more specifically the quality of the recordings used to train NN acoustic models, affects ASR performance. Even though modern ASR systems are claimed to reach human parity in conversational speech recognition[1] under certain conditions [Xiong et al., 2016], they still struggle in more challenging settings, e.g. in the presence of noise or reverberation, or for distant speech recognition [Barker et al., 2018]. A model trained with a lot of clean speech data is likely to perform very well in similar testing conditions, however, it is not a robust solution.

Mismatch between training and testing conditions, caused by speech signal variability, can substantially degrade the performance of an ASR system [Yu and Li, 2017]. Robustness of an ASR system can be enhanced by using various acoustic model adaptation methods, which aim to modify a general model towards particular testing conditions, or modify testing features towards a general model. However, as opposed to the adaptation of GMM-HMM systems, adaptation of deep neural networks still remains an open research question [Watanabe et al., 2017].

[1] However, Saon et al. [2017] found that human performance may be considerably better than previously reported.

The sources of speech variability can be of different natures and can broadly be categorised into speaker, acoustic environment, and style related. Examples of the factors contributing to speaker-related variability are speaking rate, vocal effort, regional accent, age, gender, emotional state, and anatomy of the speaker [Benzeghiba et al., 2007]. Acoustic environment variation can be caused by different noise and channel conditions; channel condition mismatch is in turn associated with different microphones and room acoustics with corresponding reverberation. Speaking style mismatch, e.g. spontaneous speech, planned monologue, or read speech, can also greatly contribute to the performance degradation.

A model adapted to particular conditions is dependent on those conditions. Although the factors contributing to the data distribution mismatch can have various sources, in this chapter we often use speaker terminology to explain the concepts, for simplicity. Speaker Independent (SI) models are trained on a large amount of recordings from different speakers, with the objective to recognise speech as well as possible for all speakers. On the other hand, Speaker Dependent (SD) models are trained for a particular speaker, and require a substantial amount of speaker-specific training data. The performance gap between SI and SD models is large [Liao, 2013; Yu and Li, 2017]. The principle underlying the Speaker Adaptive (SA) models is to adapt the SI model for a particular speaker, to match SD system performance. SA modelling, or acoustic model adaptation, is an active research direction, since it requires less speaker-specific data collection than for a SD system, is more versatile than a SD system, and exceeds the performance of a SI system.

This chapter addresses the robustness of ASR systems with respect to different sources of speech variability to compensate for the data distribution mismatch and, as a result, to improve the performance. Section 4.2.1 gives an overview of different NN-based AM adaptation techniques. Section 4.2.2 is an overview of acoustic embeddings, and Section 4.2.3 describes the embeddings used for attribute-aware and adaptive training approaches. The proposed methods and the results are presented in Section 4.3, and conclusions can be found in Section 4.5.

4.2.1 Acoustic model adaptation

In this section, we review and summarise different acoustic model adaptation approaches from the literature, to provide background for our techniques.

In general, the following adaptation approaches can be distinguished, depending on the availability of the transcripts:

- *supervised adaptation* – the gold standard transcripts for the speaker-specific adaptation data are available and used for adaptation;
- *unsupervised adaptation* – the speaker-specific adaptation data is untranscribed and labels are not used for adaptation;

Supervised adaptation is only feasible in offline scenarios. It is also costly and time-consuming. Unsupervised adaptation approaches do not depend on the adaptation data transcripts, but can give less improvements than the supervised approaches [Liao, 2013]. The methods described in this chapter do not use a disjoint adaptation dataset. We refer to the test set used in decoding as the adaptation data, and we do not use transcriptions for this set, therefore, in this sense our methods are unsupervised[2].

In this section, we adopt the classification of NN adaptation approaches from Watanabe et al. [2017], and we extend the table with a couple more recent adaptation approaches. The summary of the approaches, classified by *strategies* and *methods* is presented in Table 4.1[3].

Strategies differ by when the speaker information is used. *Test-time adaptation* strategy has three stages: training, adaptation, and testing. All or some of the model's parameters are updated during the adaptation stage, using a separate set of utterances, or test utterances with first-pass alignments (for semi-supervised approaches). At test and adaptation time, the speaker information is required, but not at training time. Test-time adaptation is often simply referred to as *adaptation*.

An *attribute-aware training* strategy incorporates the attribute information, e.g. speaker identity, into the model. Speaker labels (e.g. in adversarial training) or representations (e.g. in i-vector auxiliary inputs) are therefore necessary at both training and testing. Usually, information about the attributes is provided by a separate system, hence the performance of the attribute-aware strategies depends on the reliability of attribute information estimation.

[2]In the ASR literature, *semi-supervised training* is sometimes referred to as *unsupervised adaptation*; we use the term *unsupervised* to refer to methods independent of the adaptation transcripts.

[3]Some of the approaches can be classified as more than one method, e.g. LIN or training with i-vectors can be seen as structured paramaterisation.

Method	**Strategy**		
	Test-time adaptation	*Attribute-aware training*	*Adaptive training*
Constrained adaptation	KL div. regularisation [1]	Multi-task learning [2]	–
Feature normalistaion	LIN [3]	fMLLR [4]	fDLR [5]
Feature augmentation	–	i-vector [6] BSV [7] NaT [8]	Speaker-code [9] Auxiliary networks [10]
Structured parameterisation	LHN [11] LON [12] LHUC [13] FHL [14]	Adversarial training [15]	FHL [14] CAT [16] SAT-LHUC [17] Auxiliary networks [18] DLN [19]

Table 4.1: Classification of NN adaptation approaches adopted from Watanabe et al. [2017]. References in the table: [1] Yu et al. [2013], [2] Seltzer and Droppo [2013], [3] Neto et al. [1995], [4] Gales [1998], [5] Seide et al. [2011], [6] Saon et al. [2013], [7] Liu et al. [2014], [8] Seltzer et al. [2013], [9] Abdel-Hamid and Jiang [2013], [10] Cui et al. [2017], [11] Gemello et al. [2006], [12] Li and Sim [2010], [13] Swietojanski and Renals [2014], [14] Samarakoon and Sim [2016a], [15] Shinohara [2016], [16] Tan et al. [2015], [17] Swietojanski and Renals [2016], [18] Veselý et al. [2016], [19] Kim et al. [2017]

In *adaptive training*, the NN acoustic model consists of global, as well as attribute-specific parameters, which can be learned jointly or in a second training step. Attribute information is also usually provided externally. Speaker Adaptive Training (SAT), also referred to as SAT-DNN, has two stages – training and testing. It enables to learn attribute-specific parameters based solely on the training data, thereby reducing inference time latency. Information about the speakers has to be provided for training and testing.

Adaptation methods can broadly be categorised into *constrained adaptation*, *feature normalisation*, *feature augmentation*, and *structured parameterisation*.

Constrained adaptation focuses on introducing regularisation to avoid overfitting which can be caused by large number of AM parameters [Watanabe et al., 2017]. Kullback-Leiber (KL) divergence regularisation [Yu et al., 2013] relies on adding a KL divergence between SI and SD output distributions as a term to the

overall optimisation criterion. Weninger et al. [2019] investigate KL divergence regularisation for sequence-to-sequence ASR. In multi-task learning, an auxiliary task is introduced to improve the performance by learning shared representation. Seltzer and Droppo [2013] use acoustic state labels as the primary task, and phone labels, phone context, and state context as the secondary task. Bell and Renals [2015b] use monophone classification as a secondary task at the pretraining stage, to address context-dependent targets sparsity, and Swietojanski et al. [2015] apply a context-independent secondary task at run-time. A regressive denoising secondary task was proposed by Qian et al. [2015] to learn noise robust acoustic models. Kim et al. [2016b] generate a discriminative embedding from a bottleneck layer of a noise classification network, and use it in an attribute-aware setup: feature augmentation and multi-task learning, however, feature augmentation proved to be more effective than noise classification network used as a secondary task.

Feature normalisation methods aim to suppress the mismatch problem by normalising the input features. Feature space Maximum Likelihood Linear Regression (fMLLR) [Gales, 1998] and Feature-based Discriminative Linear Regression (fDLR) [Seide et al., 2011] are examples of input feature normalisation techniques. In fMLLR, a single affine transform per speaker is estimated with a GMM model, such that the log-likelihood that the model generates the data is maximised, based on the first-pass decode. To use this approach in a NN-based system, full GMM training is first required to transform the features, which are then used as the inputs to the NN-based acoustic model. For evaluation for unseen speakers, corresponding speaker-dependent transforms have to be estimated as a first step, to then perform second-pass decode. fDLR discriminatively estimates a fMLLR-like affine transformation of the input features by replacing the GMM system with a DNN-based one. Transforms of the input features are are modelled with a neural network and are optimised to maximise the discriminative cross entropy criterion, thus this method can be regarded as adaptive training. A test-time only adaptation normalisation method example is LIN [Neto et al., 1995], which incorporates a linear transformation layer to map speaker-dependent input features to the SI NN model. At training time, the layer can be fixed as the identity matrix or trained along with other parameters. At adaptation time, only LIN parameters are updated on a per speaker basis.

Feature augmentation methods incorporate an embedding into the model – a

compact representation of an attribute, such as speaker or noise condition. In attribute-aware training, the model does not contain attribute adaptive parameters. Speaker i-vectors [Saon et al., 2013; Senior and Lopez-Moreno, 2014] or speaker clusters i-vectors [Gupta et al., 2014] were used as fixed auxiliary inputs to acoustic models. The information about the attribute is therefore provided in the form of i-vector. The premise is that the acoustic model will be able to learn the transforms necessary to achieve invariance with respect to the attribute. Other examples of auxiliary features used for the same purpose are NaT [Seltzer et al., 2013], tandem features [Bell et al., 2013], and BSV [Liu et al., 2014]. In feature augmentation methods within an adaptive training strategy, auxiliary features are used to estimate speaker-dependent parameters. The speaker code [Bridle and Cox, 1991; Abdel-Hamid and Jiang, 2013; Xue et al., 2014] adaptation method is an example in this category, as it introduces a separate parametric space to model the speaker variability. Some recent auxiliary network approaches can also be classified in this category.

Auxiliary networks for adaptive training can generally be regarded as a way to normalise latent representations. Those networks might be learning attribute-specific parameters from the external embeddings, or internal representations. For this reason, we consider *auxiliary networks* as both *feature augmentation*, as well as *structured parameterisation* in Table 4.1. We classify our SAT-DNN as a feature augmentation method. One can argue that the neural embeddings used in this work are derived from internal representations, therefore this method might be regarded as structured parameterisation. However, the embeddings are extracted from a different acoustic model, and can provide additional information, hence we classify our auxiliary network approaches as a feature augmentation technique.

Auxiliary network adaptive training strategies, classified as feature augmentation, make the use of an extrenally extracted embedding. Miao et al. [2015] use speaker-level offline i-vectors in a SAT-DNN framework. The parameters of the auxiliary network are learned from the embedding and the activations of the last auxiliary network layer are used to shift the input features. This work is similar to one of our setups (as in Eq. 4.22), but we also experiment with (1) a different topology and complexity for the auxiliary networks $\phi_i(\boldsymbol{e}_t)$, (2) auxiliary networks trained based on different embeddings $\boldsymbol{e}_t$, (3) normalisation applied not only to the input features $\boldsymbol{x}_t$ but also to the hidden representations, and (4)

normalisation realised by shifting or scaling the general representations.

Delcroix et al. [2015] propose a context adaptive DNN model, with one or several layers to learn factor-specific parameters. The output of the factorised layer is obtained by weighted averaging over the contribution of the different factor classes, given posteriors over the factor classes. Two factors corresponding to gender are explored in this study. The posteriors are obtained from clustering the i-vectors and represent the acoustic context. Delcroix et al. [2018] is an extension of this work, where the context class weights are derived from the auxiliary features, by transforming them through an auxiliary network. The context networks realise gating to adapt all parameters of a layer. The auxiliary classes used in this work are genders and speakers. Only a single layer is factorised, and utterance-level offline i-vectors are used to realise rapid adaptation.

Zhao et al. [2018] perform the anchor-based speaker adaptation with the adaptation network using d-vectors (described in Section 4.2.2) and Low-Rank Plus Diagonal (LRPD) transformation method. Adapting multiple layers (all layers in a 6 layer DNN with 2048 units) gave 18% relative reduction over the SI baseline trained with 3,400h of data. Adapting multiple layers was better than adapting a single layer - the recognition accuracy improved continuously as the number of the adapted layers increased until the last hidden layer. However, for a smaller dataset (220h), concatenation of the d-vectors to the input resulted in the same relative reduction in WER (18%) compared to the matching SI baseline. It is unclear from the paper if this gain will hold for the larger scale task. The training procedure in the paper is two-stage: train the main part of the network, keep it fixed, then train the adaptation part of the network.

In [Cui et al., 2017] speaker-dependent mappings are generated by a control network from speaker level i-vectors. Unlike in [Zhao et al., 2018], the main and control networks are jointly optimised. In the Babel task, the main model is a 5 layer DNN, with one layer removed when applying speaker adaptation with a control network. A control network is inserted after every hidden layer. In Callhome and Switchboard tasks, however, the adaptation is applied only to one or two bottom layers of the main network, and the main network is a 5 layer bi-directional LSTM. The control network consists of 3-4 shared fully-connected layers and layer-dependent elementwise affine transformations (bias and scaling layers). The embedding-based adaptation is performed on top of the speaker-aware training (appending i-vectors to the input). The improvements over the

speaker-aware model are 0.1-0.4% absolute for Callhome and Switchboard.

For real-time decoding, online i-vectors extracted at a frame level can be used in an attribute-aware strategy. Ochiai et al. [2017] propose an alternative to online i-vectors for this task – cumulative moving averaged bottleneck speaker vectors. The frame-wise embeddings are used to adapt a CNN model in a streaming fashion, by factorising a single convolutional layer into two sub-layers. The auxiliary network is used to calculate interpolation coefficients of each factorised sub-layer, based on the embeddings. Experiments for CHIME-3 show similar performance of the proposed embeddings, compared to online frame-wise i-vectors.

Other normalisation techniques are also related to our work. Yoo et al. [2019] generate scaling and shifting parameters, with internal or external representations, for multi-dialect ASR. Using the external representation is equivalent to embedding-based auxiliary networks, whereas the use of the internal representations is equivalent to auxiliary networks in a structured parameterisation scheme. Conditioning on the external representation is also referred to as Feature-wise Linear Modulation (FiLM) and it was recently proposed for visual tasks [Perez et al., 2018]. In FiLM, each feature (or channel for a CNN model) is scaled and shifted by FiLM parameters. The FiLM layer is defined as:

$$FiLM(\boldsymbol{x}) = \gamma(\boldsymbol{z}) \odot \boldsymbol{x} + \beta(\boldsymbol{z}) \tag{4.1}$$

where γ and β are $\boldsymbol{z}$-dependent scaling and shifting vectors, $\boldsymbol{z}$ is an external conditioning representation, and $\boldsymbol{x}$ is the layer's input vector [Dumoulin et al., 2018]. To match the notation used in this chapter, the activations of the layers with FiLM can be formulated as:

$$\begin{aligned} \boldsymbol{a}_t^i &= \sigma(\boldsymbol{W}_i^{\mathsf{T}} \hat{\boldsymbol{x}}_t^i + \boldsymbol{b}_i) \qquad\qquad 1 \leq i \leq L \\ \hat{\boldsymbol{x}}_t^i &= \gamma_i(\boldsymbol{e}_t) \odot \boldsymbol{x}_t^i + \beta_i(\boldsymbol{e}_t) \end{aligned} \tag{4.2}$$

where $\hat{\boldsymbol{x}}_t^i$ is the normalised (FiLM-ed) input to layer i at time step t. $\gamma_i(x)$ and $\beta_i(x)$ are analogous to our scaling weight layer $g_i(x)$ and bias weight layer $h_i(x)$ in Eq. 4.21. The main difference between the formulation of the two approaches lies in the additional SI term used in our approach, implemented as a skip connection (see Eq. 4.20). Yoo et al. [2019] use two shared layers (analogous to $f(x)$) to transform the dialect embedding (one-hot vector in this work), and *tanh* activation functions for $\gamma_i(x)$ and $\beta_i(x)$ to adapt LSTM layers.

Feature augmentation methods can also be viewed as a special case of structured parameterisation. For attribute-aware training, only the first layer bias vector is adapted [Watanabe et al., 2017], whereas all hidden layers can be adapted with adaptive training feature augmentation approaches. Nevertheless, we keep the discrimination into feature augmentation and structured parameterisation to distinguish the methods which make the use of an external embedding augmenting the input features.

Structured parameterisation imposes adaptable structures on the DNN hidden layers. The activations for structured parameterisation methods can be generally defined as:

$$\boldsymbol{a}_t^i = \psi(\boldsymbol{W}_i^{s\mathsf{T}} \boldsymbol{x}_t^i + \boldsymbol{b}_i^s) \qquad 0 \leq i \leq L \qquad 0 \leq s \leq S \tag{4.3}$$

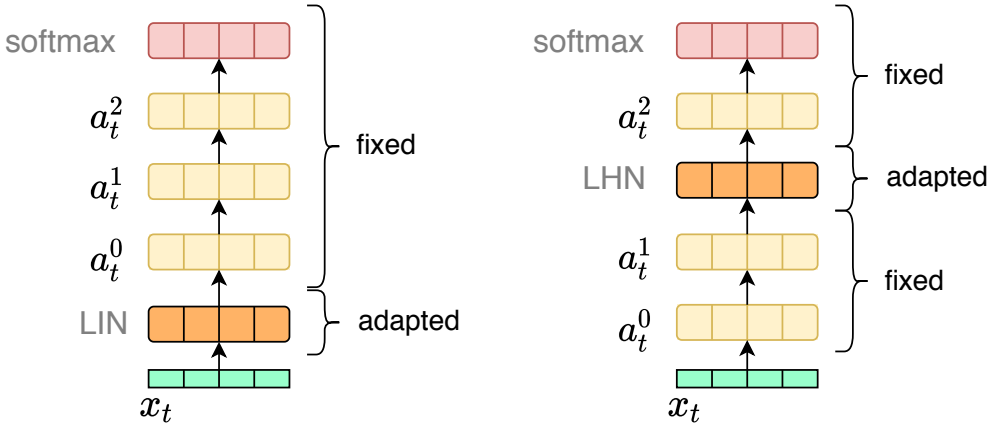
where $\boldsymbol{W}_i^s$ and $\boldsymbol{b}_i^s$ are speaker-dependent weight matrix and bias vector for each speaker s. A subspace for speaker adaptation is separated from the phonetic modelling space by factorising the transformations in hidden layers into global and speaker-dependent transformations.

LIN [Neto et al., 1995], LHN [Gemello et al., 2006] and LON [Li and Sim, 2010] are linear transformations used typically at test time. For LIN and LHN (Figure 4.1), an additional linear layer is inserted into the model and therefore the speaker-dependent transformation matrix and bias vector are given by

$$\begin{aligned} \boldsymbol{W}_i^s &= \boldsymbol{W}_i \boldsymbol{A}_i^s \\ \boldsymbol{b}_i^s &= \boldsymbol{W}_i \boldsymbol{k}_i^s + \boldsymbol{b}_i \end{aligned} \tag{4.4}$$

where $\boldsymbol{A}_i^s$ and $\boldsymbol{k}_i^s$ are weight matrix and bias of the additional linear transform layer (LIN or LHN). $\boldsymbol{W}_i$ and $\boldsymbol{b}_i$ are global weight matrix and bias vector, respectively. For LON, SD transformation $\boldsymbol{A}_i^s$ is represented using a linear interpolation of the set of basis matrices. The Cluster Adaptive Training (CAT) is another linear-basis interpolation approach, which uses full rank adaptation bases to model the SD transformation [Tan et al., 2015, 2016, 2018; Wu and Gales, 2015]. In CAT, a linear interpolation of the cluster Gaussian components means is used as the mean for each speaker.

Methods that adopt a linear-basis interpolation structure model the transfor-



(a) LIN (b) LHN

Figure 4.1: Linear Input Network (LIN) and Linear Hidden Network (LHN) examples with three hidden layers. During adaptation, all but LIN/LHN layer weights are fixed.

mation in each layer as:

$$\boldsymbol{W}_i^s = \boldsymbol{W}_i + \sum_{n=1}^{N} \boldsymbol{\alpha}_{i,n}^s \boldsymbol{W}_{i,n} \qquad \boldsymbol{b}_i^s = \boldsymbol{b}_i + \sum_{m=1}^{M} \boldsymbol{\beta}_{i,m}^s \boldsymbol{b}_{i,m} \tag{4.5}$$

where the speaker-dependent weight matrix $\boldsymbol{W}_i^s$ is decomposed into a global weight matrix $\boldsymbol{W}_i$ and SD linear interpolation of the set of basis matrices, $\boldsymbol{W}_{i,n}$. Similarly, the bias vector is decomposed into a global vector $\boldsymbol{b}_i$ and basis vectors $\boldsymbol{b}_i^m$. $\boldsymbol{\alpha}_{i,n}^s$ and $\boldsymbol{\beta}_{i,m}^s$ are speaker-dependent interpolation weights. The number of bases can be adjusted to control the number of speaker-dependent parameters. To reduce the number of speaker-dependent parameters and avoid overfitting Swietojanski and Renals [2014] and Swietojanski et al. [2016] propose LHUC. The transformation matrix in LHUC is constrained to be diagonal and there is no speaker-dependent bias vector. The transformations in the layers become:

$$\begin{aligned} \boldsymbol{W}_i^s &= \boldsymbol{W}_i \boldsymbol{A}_i^s \qquad \boldsymbol{A}_i^s = diag(\alpha_{i,n}^s) \\ \boldsymbol{b}_i^s &= \boldsymbol{b}_i \end{aligned} \tag{4.6}$$

where $\alpha_{i,n}^s \in [0, 2]$ is a speaker-dependent scaling factor. LHUC is a test-only adaptation method, however, it can be extended to SAT-LHUC for an adaptive training strategy [Swietojanski and Renals, 2016].

Samarakoon and Sim [2016a] propose Factorised Hidden Layer (FHL) for speaker-dependent parameter reduction. FHL models SD layers by representing

an SD affine transformation as a linear combination of multiple rank-1 bases[4]:

$$\begin{aligned} \boldsymbol{W}_i^s &= \boldsymbol{W}_i + \boldsymbol{A}_i^s \qquad \boldsymbol{A}_i^s \approx \boldsymbol{U}_i \Sigma_i^s \boldsymbol{V}_i \mathsf{T} \\ \boldsymbol{b}_i^s &= \boldsymbol{b}_i + \boldsymbol{B}_i^\mathsf{T} \boldsymbol{k}_i^s \end{aligned} \tag{4.7}$$

where the speaker-dependent transformation matrix $\boldsymbol{A}_i^s$ can be approximated with a singular value decomposition with rank-1 matrices, and $\boldsymbol{B}_i$ is a weight matrix used to transform the speaker-dependent bias vector. $\boldsymbol{k}_i^s$ is the SD interpolation vector. The formulation of the speaker-dependent bias $\boldsymbol{b}_i^s$ is similar to the effective bias term in Eq. 4.18. The interpolation vectors in FHL are initialised as speaker offline i-vectors, therefore the transformation $\phi_i(x)$ is speaker-dependent. The activations of the FHL model are:

$$\begin{aligned} \boldsymbol{a}_t^i &= \psi((\boldsymbol{W}_i + \boldsymbol{A}_i^s)^\mathsf{T} \boldsymbol{x}_t^i + \boldsymbol{b}_i + \boldsymbol{B}_i^\mathsf{T} \boldsymbol{k}_i^s) \\ &= \psi(\boldsymbol{W}_i^\mathsf{T} \boldsymbol{x}_t^i + \boldsymbol{b}_i) + \phi_i(x) \\ \phi_i(x) &= \psi(\boldsymbol{A}_i^{s\mathsf{T}} \boldsymbol{x}_t^i + \boldsymbol{B}_i^\mathsf{T} \boldsymbol{k}_i^s) \end{aligned} \tag{4.8}$$

Examples of works using auxiliary networks for structured parameterisation include [Veselý et al., 2016; Delcroix et al., 2018; Sarı et al., 2019]. Veselý et al. [2016] use a sequence summarising auxiliary network for speaker adaptation. An acoustic summary of an utterance is extracted as the average of the activations of an auxiliary network. The performance of those embeddings was compared with i-vectors, and they gave similar results for the AMI task. Delcroix et al. [2018] used a similar approach in the context of end-to-end ASR. A sequence summary auxiliary network was used to project the representations at the encoder's input. In [Sarı et al., 2019] speaker offsets are learned to adapt LSTM layers of the main network. The offsets are automatically generated by the auxiliary network, however, speaker or phonetic level averages of the activations are used instead of utterance summarisations. The embeddings are the targets for the auxiliary network, making this approach similar to multi-task learning, but applied to the lower layers of the network. The generated offsets perform only shifting (not scaling), and only one hidden layer is adapted in this way.

Recently, Layer Normalisation (LN), described previously in Section 2.2.2, was proposed by Ba et al. [2016] to normalise the activations of a NN model for training speed and stability improvements. Kim et al. [2017] used Dynamic

[4]The combination weights are initialised with i-vectors and can be refined at test-time. For this reason, FHL can also be regarded as a form of feature augmented adaptive training, since it incorporates the i-vector representation into the model.

Layer Normalisation (DLN) for acoustic model adaptation, which can be seen as structured parameterisation. To enhance the invariance to different speakers and environments, an utterance summarisation is used to generate scaling and shifting parameters and use them for normalisation as in LN. Utterance summarisation vector is extracted in each layer of the model as segment-level mean of the activations of the previous layer, and used to dynamically generate the parameters for normalisation for that layer[5]. Hence, the auxiliary embeddings are not used in this approach. The utterance summarisation vector extractor is trained jointly with the acoustic model, and the speaker information is not needed in this method. This is an adaptive training strategy; the adaptation parameters are not fine-tuned on the adaptation data – they are dynamically generated from summarisations of the utterances extracted from the input features at training and at test time. DLN was also explored by Yoo et al. [2019] for multi-dialect speech recognition.

Adversarial training [Goodfellow et al., 2014] can be seen as attribute-aware structured parameterisation. Shinohara [2016] proposed adversarial training for speech recognition to enhance the noise invariance of learned representations. As in the multi-task approach, a shared representation is learned, however, it is learned adversarially to the secondary task, by reversing the gradient in back-propagation for the secondary task. Various adversarial task have been proposed. Low domain (noise type) classification accuracy is induced by Shinohara [2016], Serdyuk et al. [2016] and Guo et al. [2019]. Speaker classification loss is used as a secondary task in [Meng et al., 2018, 2019a,b] to enhance speaker-invariant representations. Sun et al. [2018] use domain adversarial training for accented speech recognition, and Hu et al. [2019] apply this approach to multilingual acoustic modelling, to learn language-invariant features.

NN adaptation approaches can also be categorised as *model-* or *feature-space*. *Model-space* DNN adaptation consists in direct update of the parameters of the model. *Feature-space* adaptation involves the estimation of feature space transforms, i.e. is equivalent to input feature normalisation methods. Also, the auxiliary inputs methods (e.g. i-vectors, BSV, NaT) can be seen as feature-space adaptation, as they normalise the inputs to the acoustic model. Feature- or model-space adaptation approaches can be applied atop of SAT-DNNs – the pa-

[5]We define a *segment* or an *utterance* as a short (a couple of seconds) piece of speech. We use the terms *segment* and *utterance* interchangeably throughout the thesis.

rameters are optimised on the training set and fine-tuned on the adaptation data. For example, it was shown by Samarakoon and Sim [2016b] that updating the i-vectors with the use of adaptation data can bring improvements over adaptively trained models. Similarly, model-space adaptation can also further improve the recognition accuracy of SAT-DNNs [Miao et al., 2015; Swietojanski et al., 2016; Samarakoon and Sim, 2016a].

We focus on unsupervised attribute-aware and adaptive training. Unlike test-time adaptation, the methods do not require the SD parameters to be learned at test time, thus reducing the latency of acoustic model predictions and making them more suitable for real-time or low-latency ASR. Methods proposed in this chapter can be classified as feature augmentation and structured parameterisation. We rely on the learned embeddings to normalise the nuisance factors in training and testing.

4.2.2 Acoustic embeddings

In this section, we describe the most commonly used acoustic embeddings of utterances and speakers.

i-vectors

i-vectors were proposed by Dehak et al. [2009]; Dehak et al. [2011] for the task of speaker verification and recognition. i-vectors are low-dimensional embeddings, where the speaker and channel subspace is modelled jointly in a “total variability" subspace $\boldsymbol{T}$. A Universal Background Model (UBM) model, which is a GMM with K Gaussian compnents, is needed for the i-vector extraction. The t-th frame $\boldsymbol{x}_t$ from the u-th speech utterance (segment) is generated from the UBM model as [Miao et al., 2015]:

$$\boldsymbol{x}_t \sim \sum_{k=1}^{K} \gamma_t(k) \mathcal{N}(\boldsymbol{\mu}_k + \boldsymbol{T}_k \boldsymbol{w}_u, \boldsymbol{\Sigma}_k), \tag{4.9}$$

where $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ are mean vector and covariance matrix for k-th Gaussian component, the total variability matrices $\boldsymbol{T}_k$ span a subspace of the shifts by which the UBM means are adapted to particular segments, $\gamma_t(k)$ is the posterior probability of the k-th Gaussian given the speech frame. The latent vector $\boldsymbol{w}_s$ has a standard normal distribution and corresponds to the coordinates of the mean

shift in the total variability subspace $\boldsymbol{T}_k$. The estimate of vector $\boldsymbol{w}_u$ is the i-vector and can be realised with MAP.

To extract the i-vectors, first the UBM is trained using ML, to obtain the parameters $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$, and $\gamma_t(k)$. Then, to train matrix $\boldsymbol{T}_k$, sufficient statistics are accummulated for each speech frame within a segment s of length T as:

$$n_k(u) = \sum_{t=1}^{T} \gamma_t(k), \tag{4.10}$$

$$\boldsymbol{f}_k(u) = \sum_{t=1}^{T} \gamma_t(k)\boldsymbol{x}_t. \tag{4.11}$$

n_k is the zeroth order Baum-Welch statistic, f_k is the first order statistic. $\boldsymbol{T}_k$ is updated from the above statistics iteratively for all training utterances.

Assuming the prior distribution $p(\boldsymbol{w}_u)$ is standard normal, the i-vector can be extracted as [Vestman and Kinnunen, 2018]:

$$p(\boldsymbol{w}_u|\boldsymbol{X}_u, \boldsymbol{T}) = \mathcal{N}(\boldsymbol{\mu}_u, \boldsymbol{\Sigma}_u), \tag{4.12}$$

$$\boldsymbol{\Sigma}_u = \left(\boldsymbol{I} + \sum_{k=1}^{K} n_k(u)\boldsymbol{T}_k^T \boldsymbol{\Sigma}_k^{-1} \boldsymbol{T}_k \right)^{-1}, \tag{4.13}$$

$$\boldsymbol{\mu}_u = \boldsymbol{\Sigma}_u \sum_{k=1}^{K} \boldsymbol{T}_k^T \boldsymbol{\Sigma}_k^{-1} (\boldsymbol{f}_k(u) - n_k(u)\boldsymbol{\mu}_k), \tag{4.14}$$

where $\boldsymbol{X}_u = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T\}$ is a sequence of all feature vectors for utterance u. Mean $\boldsymbol{\mu}_u$ of the posterior i-vector distribution is the i-vector of utterance u.

i-vectors are linear utterance or speaker embeddings – a speaker i-vector is an average of utterance i-vectors for that speaker. A number of different approaches have been proposed to incorporate non-linear processing into the i-vector extraction framework for improved speaker discriminability. [Lei et al., 2014] show that replacing the GMM with a DNN to collect sufficient statistics can bring the improvements in the i-vector based speaker recognition task. Snyder et al. [2015] explore supervised full-covariance GMM for this task, derived from TDNN posteriors, as an alternative to DNN-based UBM, with lower computational complexity. Other examples of speaker or utterance embeddings in this section are NN-based alternatives to i-vectors.

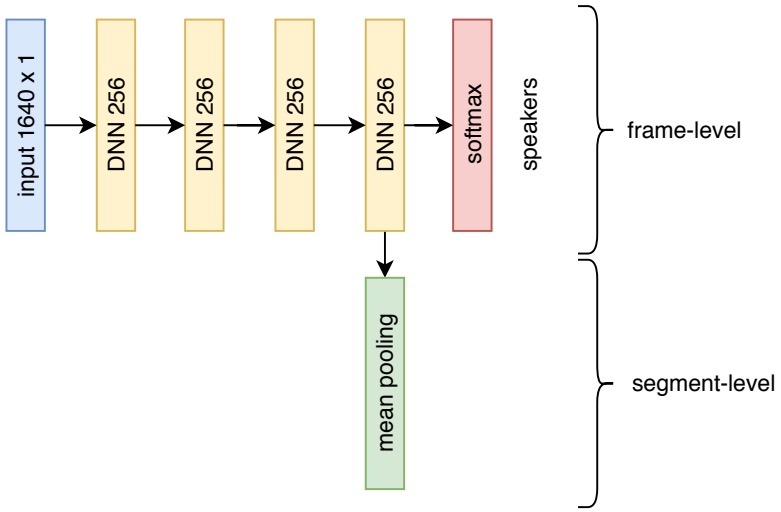


Figure 4.2: d-vector extraction framework. Input is 40D FBANKs with 30 frames of context to the left and 10 to the right. The layers are fully-connected. The mean pooling component compute segment-level mean of the activations from the last hidden layer. Mean pooling was employed to capture the characteristics of the whole segment. Embedding extraction component is marked in green.

d-vectors

Variani et al. [2014] propose a d-vector, extracted as illustrated in Figure 4.2. The network is trained for a speaker classification task. The inputs are frame-wise feature vectors (FBANKs) with a larger context than in a typical acoustic model. All layers are fully-connected. The mean pooling component is averaging the frame-wise activations over a segment. For a speaker embedding, segment embeddings are averaged. d-vectors have proved to be complementary to i-vectors and more robust to additive noise in speaker verification.

x-vectors

x-vectors (Figure 4.3) were proposed by Snyder et al. [2017] for speaker verification. Like d-vectors, they use a NN for the embedding extraction, however the pooling component is integrated into the network. The statistics pooling layer aggregates the frame-level representations over the input segment by computing segment-level mean and standard deviation of the activations from the previous layer. The statistics vectors are concatenated and passed to two additional layers before the softmax classification output layer. x-vectors are typically extracted as the activations from the first hidden layer after the statistics pooling

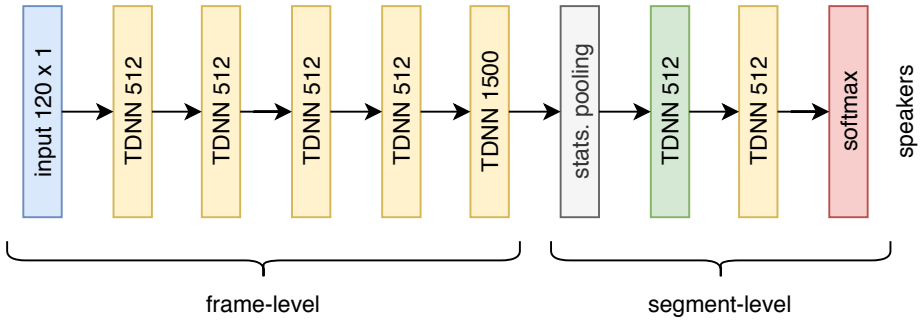


Figure 4.3: x-vector extraction framework. Input is MFCCs spliced across 5 frames plus delta and delta delta features. The layers are TDNNs, i.e. 1D conv layers with dilation). Embedding extraction component is marked in green.

layer. For speaker x-vectors, utterance x-vectors are averaged. Like i-vectors and d-vectors, x-vectors are low-dimensional (512 dimensions) utterance or speaker embeddings. Snyder et al. [2018] have shown that they can effectively exploit data augmentation and outperform the i-vector baseline for Speaker Verification (SV). A similar approach was proposed by Shon et al. [2018], where a CNN model with 1D convolutions is used for the embedding extraction.

Zhu et al. [2018] propose self-attentive x-vectors, where the embedding is a weighted average of frame-level activations. The general architecture is the same as for the x-vectors, but the statistics pooling layer is replaced with a self-attention layer to derive a weighted mean and standard deviation from the previous frame-wise hidden layer. The weights are trained within the system to maximise speaker classification performance. The effect of multiple attention head is investigated, to model speakers discriminability along multiple aspects. Self-attention with multiple attention heads was found to be superior to standard x-vectors for speaker verification. Okabe et al. [2018] also incorporate attentive statistics pooling into x-vector extraction and also show the improvements.

Other deep speaker embeddings

A complete literature review on deep speaker embeddings is beyond the scope of this thesis. However, since we explore the role of different types of the embeddings on adaptation performance and we propose an AM utterance embedding derived from a deep CNN model for this task, we will give a brief overview of some of the most recent deep speaker embeddings, with the focus on deep CNN embedding extractors with 2D convolutions in time and frequency (in contrast to x-vectors

which use convolutions in time only).

Currently, variants of VGG [Simonyan and Zisserman, 2015] or ResNet [He et al., 2016] architectures are used for the speaker embedding extraction. The methods differ mainly by aggregation strategies and by different loss functions used in training.

Chung et al. [2018] use average pooling layer to obtain the utterance-level embedding. Bhattacharya et al. [2017] employ attention to assign weights to frame-wise representations. Other alternative aggregators, NetVLAD [Arandjelovic et al., 2016] and GhostVLAD [Zhong et al., 2018], are explored by Xie et al. [2019] for speaker embedding extraction. Those layers can be trained discriminatively within the network, hence are well suited for end-to-end solutions.

Alternative loss functions have been proposed to enforce intra-class compactness and inter-class diversity. Nagrani et al. [2017] use a VGG-like architecture with softmax loss for identification and contrastive softmax loss within a Siamese network for verification. Chung et al. [2018] explore contrastive softmax loss within a ResNet architecture. Xie et al. [2019] also use a ResNet architecture, but explore Additive Margin softmax (AM-softmax) loss. Hajibabaei and Dai [2018] compare Angular softmax (A-softmax) with AM-softmax, and propose Logistic Margin loss. Angular softmax introduces a margin between target and non-target class into the loss function. We refer the reader to the referenced papers for more details.

Nagrani et al. [2019] give an overview of different aggregation across time strategies and different loss functions. The authors provide a comparison of the results for the VoxCeleb dataset for different architectures, aggregation approaches (global average pooling, NetVLAD, GhostVLAD), and loss functions (standard softmax, large-margin softmax and the contrastive loss). Chung et al. [2020] evaluate some of the most recent loss functions within controlled experimental conditions (softmax, AM-softmax, AAM-softmax), compare them with metric learning objectives (triplet, generalised end-to-end, prototypical), and propose a new variant of angular prototypical objective, showing its superior performance for VoxCeleb. We refer the reader to those papers for a systematic comparison of different aggregation strategies and loss functions.

Bhattacharya et al. [2019] achieve state-of-the-art performance on VoxCeleb with a ResNet architecture, self-attentive pooling and large-margin loss function. They identify feature normalisation as a key factor contributing to the

performance gains. Zhou et al. [2019] integrate the phonetic information into the speaker embedding extraction. They use a RestNet architecture with a multi-head attentive pooling layer for temporal aggregation and L2-constrained softmax loss [Ranjan et al., 2017]. The phonetic information is extracted from a pre-trained acoustic model as bottleneck features and combined with the input FBANKs or at the attentive pooling layer. Speaker verification performance gains are shown with the use of phonetic features for the VoxCeleb dataset.

4.2.3 Embeddings for acoustic model adaptation

Various utterance embeddings can be used for acoustic model adaptation. *Offline* i-vectors are the representations extracted at the speaker or utterance level, imposing the availability of speaker or utterance data prior to modelling the acoustics. Speaker-level offline i-vectors have been investigated in the attribute-aware training strategy by Saon et al. [2013]. Utterance-level offline i-vectors were investigated by Senior and Lopez-Moreno [2014].

Unlike offline i-vectors, *online* i-vectors are causal in nature. The sufficient statistics for the i-vector extraction are accumulated and carried over. Depending on how frequently the sufficient statistics are updated, online i-vectors can be segmental or per frame. For frame-level updates the statistics can be accumulated within utterances or within speakers, and the i-vectors are extracted at the frame level. For segmental online i-vectors, the statistics are accumulated within speakers, and the i-vector extraction is at the utterance level, i.e. the i-vector is constant across all the frames of a given segment. The classification of the i-vector types used for AM is presented in Figure 4.4.

Segmental (utterance-level) online i-vectors were previously investigated for attribute-aware and adaptive training by Garimella et al. [2015]. Here, the online i-vectors are estimated for each utterance, taking into account the past utterances for a given speaker. To bias the sufficient statistics to the most recent history, they apply an exponential decay. The authors use HMM state alignment information from an ASR system for accumulating the sufficient statistics – a single Gaussian per HMM state replaces the UBM-GMM for estimating i-vectors. Utterance-level online i-vectors provide more variability in training, giving more robustness to unseen testing conditions.

Online frame-wise i-vectors are better suited for real-time ASR. Peddinti et al.

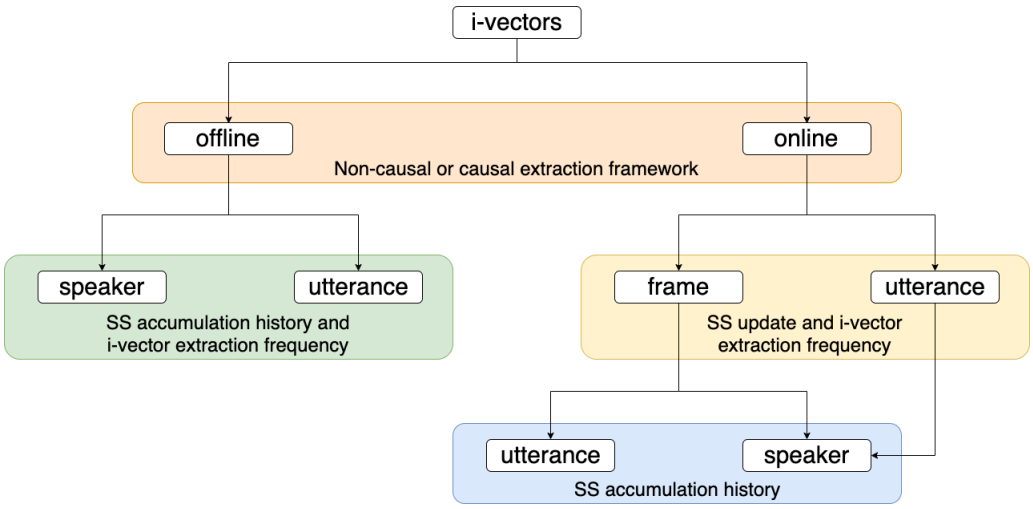


Figure 4.4: Classification of the i-vectors used in ASR for AM adaptation.

[2015a] extract online frame-level i-vectors using only the frames prior to the current frame, to address recognition for unsegmented audio recordings. Arsikere and Garimella [2017] also use frame-level online i-vectors to address attribute-aware training with frequent speaker changes, and the statistics are derived from the DNN AM posteriors, using the top K posteriors scores to mitigate the effect of incorrect alignments. Arsikere et al. [2019] use frame-level online i-vectors for multi-dialect acoustic modelling.

Except for the papers mentioned in this section, most of the previous works on i-vector based AM adaptation rely on offline i-vectors.

A number of other representations have previously been investigated for attribute-aware and adaptive training. The goal of those approaches is to compensate for the variability caused by different speech attributes, as previously discussed. Seltzer et al. [2013] use a noise estimate vector (fixed over the utterance), and Kundu et al. [2016] extract noise-dependent bottleneck features for feature augmentation. Room characteristics were also extracted as bottleneck features from the distance-discriminant DNN and used as auxiliary features for far-field ASR by Miao and Metze [2015]. Factorised i-vectors were proposed by Karanasou et al. [2014] to learn independent speaker and noise representations by constraining them to be orthogonal. This approach proved to be more effective than using separately trained speaker and noise i-vectors for adaptation.

Alternatively, a single representation that encodes multiple attributes can be used for adaptation [Watanabe et al., 2017]. Kundu et al. [2016] use a multi-attribute bottleneck DNN classification network to encapsulate multiple

attributes in a single embedding – speaker, noise and phones. Veselý et al. [2016] propose a sequence-summarising auxiliary network to extract the representation of the utterance, without any prior knowledge about the attributes, and use it for adaptation. Similarly, Tran et al. [2017] extract auxiliary features from the acoustic model activations and feed it back to the input layer for adaptation. Delcroix et al. [2018] use sequence summary for an end-to-end AM model adaptation.

4.3 Embedding-based attribute-aware and adaptive training

This section is based on [Rownicka et al., 2017], [Rownicka et al., 2018] and [Rownicka et al., 2019]. We first introduce the proposed Acoustic Model (AM) embeddings, and then we describe our methods for embedding-based attribute-aware and adaptive training.

For attribute-aware training we use Aurora-4, MGB-3, Financial and Political News and AMI IHM datasets. Aurora-4 is used to evaluate the adaptation performance on noisy data – with additive noise and recorded with mismatched microphones. Similarly for MGB-3, since this dataset is also noisy, but also larger and more challenging than Aurora-4. We use Financial and Political News data mainly for real-time and low latency decoding scenarios. The goal of evaluation on AMI IHM is to ensure that the embedding-based attribute-aware training is also effective when differences between speakers are the main source of variability. Those results also serve as a benchmark for adaptive training experiments, where we focus on speaker adaptation and evaluate on AMI IHM dataset.

4.3.1 Acoustic model embeddings

We propose Acoustic Model (AM) embeddings as utterance summaries derived from a deep CNN or a DNN acoustic model, trained to output posterior probabilities of senones [Rownicka et al., 2018]. Figure 4.5 shows the extraction framework for a deep CNN model. The motivation to extract AM embeddings is two-fold. First, we extract them to facilitate the analyses of learned representations, by encapsulating the representations from different layers in a single embedding. We refer to this embedding as a whole model embedding in Chapter 6, where we present the analyses which aim to better understand the representations learned

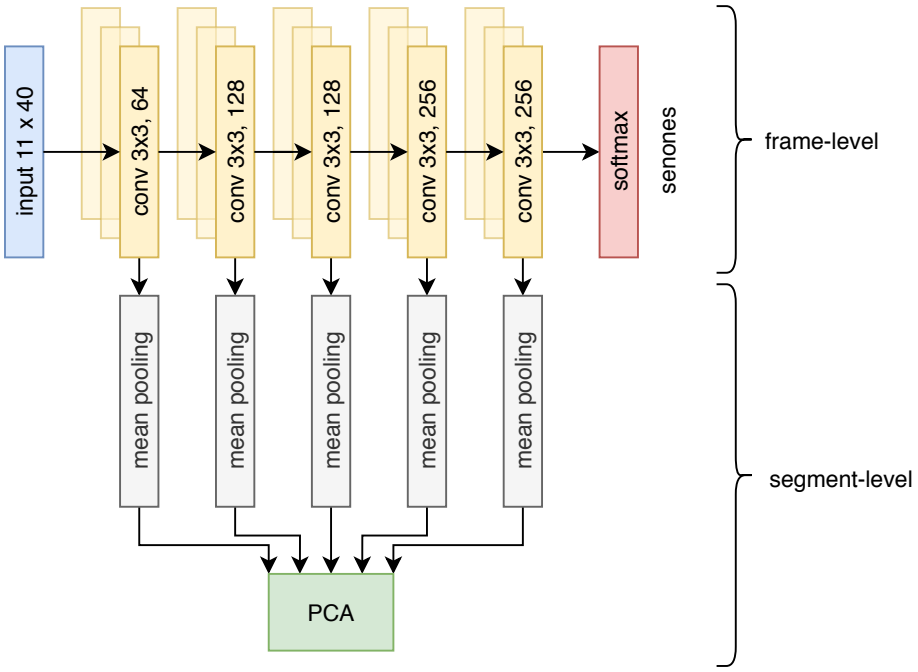


Figure 4.5: Deep CNN utterance embedding extraction framework. The mean pooling components compute segment-level mean of the activations from selected layers. Embedding extraction component is marked in green.

by well performing acoustic models. Secondly, we evaluate the performance of the utterance embeddings for acoustic model adaptation, motivated by the fact that the learned representations can be regarded as vectors summarising the acoustics in the utterance. We compare AM embeddings with i-vectors and x-vectors, and we present the results for attribute-aware and adaptive training later in this chapter. We also relate the information encoded in the embeddings to their adaptation performance in Chapter 6.

To extract the utterance-level embeddings, we perform a forward pass of the frame-level 40-dimension FBANK features using ± 5 frames of context, with mean and variance normalization, through trained acoustic models. The deep CNN consists of fifteen convolutional layers, with three layers in each convolutional block[6] (using 3x3 kernels), with the number of channels doubling after each convolutional block (Figure 4.5) [Rownicka et al., 2017]. The DNN embedding extractor is a six layer network with 2048 nodes in each layer. To extract the CNN embedding, we average the frame level output of each convolutional block across utterance (mean pooling) before applying the ReLU function so that the distribution is closer to

[6] A block consists of three convolutional layers of the same structure.

normal, and hence better suited for further processing. We then vectorise and concatenate pooled segment-level representations, merging the information across layers. For the deep CNN model this results in a 35k-dimensional vector, and for the DNN a 12k-dimensional vector – the activations are pooled from each hidden layer. Then, a joint PCA transform is used to find the directions of highest variation across the network and to reduce the dimensionality of the final embedding.

The proposed embeddings extracted from the acoustic models are fixed-dimensional utterance summaries. Unlike in the x-vector approach, speaker labels are not used in the AM embedding extraction. They can be seen as multi-attribute utterance embeddings, because they are designed to summarise an utterance, without imposing any specific speech attribute representation. Moreover, unlike Veselý et al. [2016], in our work segment averaging is not in the final component – the embeddings combine different views on the utterance (i.e. different levels of abstraction) by merging the representations across selected layers of a model. We hypothesise that this extraction framework enables to capture acoustic characteristics of an utterance well and can be useful for adaptation. In Chapter 6 we evaluate which characteristics can be captured and how they relate to the adaptation performance. In this chapter, we seek to optimise embedding-based SAT-DNNs, and we use AM embeddings for this task.

4.3.2 Normalisation of hidden representations

Both attribute-aware and adaptive training methods described in this chapter can be seen as normalisation techniques. In its simplest form, SI DNN acoustic model consists of multiple hidden layers and a softmax classification layer (see Section 2.2.4 for the formulation of alternative general AM architectures). The activations of the i-th layer are defined as:

$$\boldsymbol{a}_t^i = \sigma(\boldsymbol{W}_i^{\mathsf{T}} \boldsymbol{x}_t^i + \boldsymbol{b}_i) \qquad 0 \leq i \leq L \tag{4.15}$$

where L is the total number of layers, the weight matrix $\boldsymbol{W_i}$ connects i-th layer with a previous one, and $\boldsymbol{b_i}$ is the bias vector for layer i. $\sigma(x)$ is the non-linearity (i.e. the activation function). We mostly use the $ReLU$ activation function in the experiments, $\sigma(x) = \max(0, x)$, except for the last layer which uses the softmax function. We denote the input feature vector at the t-th time step as $\boldsymbol{x}_t$. The

inputs to the i-th layer at each time step t ($\boldsymbol{x}_t^i$) are therefore

$$\boldsymbol{x}_t^i = \begin{cases} \boldsymbol{x}_t & i = 0 \\ \boldsymbol{a}_t^{i-1} & 1 \leq i \leq L \end{cases} \tag{4.16}$$

Attribute-aware training can be realised by concatenating a frame-wise embedding $\boldsymbol{e}_t$ with the input features. Hence, the inputs to the i-th layer become[7]:

$$\boldsymbol{x}_t^i = \begin{cases} [\boldsymbol{x}_t \quad \boldsymbol{e}_t] & i = 0 \\ \boldsymbol{a}_t^{i-1} & 1 \leq i \leq L \end{cases} \tag{4.17}$$

The matrix-vector product, obtained by substituting $\boldsymbol{x}_t^1$ to the Equation 4.15, can be decomposed into two vector-matrix sub-products

$$\boldsymbol{W}_1^\mathsf{T}[\boldsymbol{x}_t \quad \boldsymbol{e}_t] = \boldsymbol{A}_1^\mathsf{T}\boldsymbol{x}_t + \boldsymbol{B}_1^\mathsf{T}\boldsymbol{e}_t,$$

therefore, the concatenation of the vectors is equivalent to shifting the first hidden layer representations. The activation of the first hidden layer becomes

$$\boldsymbol{a}_t^1 = \sigma(\boldsymbol{A}_1^\mathsf{T}\boldsymbol{x}_t + \boldsymbol{B}_1^\mathsf{T}\boldsymbol{e}_t + \boldsymbol{b}_1) \tag{4.18}$$

where the effective bias is $\boldsymbol{B}_1^\mathsf{T}\boldsymbol{e}_t + \boldsymbol{b}_1$. This effective bias is embedding-dependent, and its role in the AM training is to reduce the variability caused by nuisance attributes, from the point of view of modelling posterior probabilities of CD HMM states given the augmented input, i.e. perform some normalisation of the first hidden layer.

In the adaptive training, additional attribute-specific parameters are introduced into the model. One possible approach is to use an auxiliary network $\phi(x)$ to further transform the effective bias to the first hidden layer such that it is better suited for the normalisation task. Auxiliary networks can also be used for the normalisation of each SI hidden layer i of the acoustic model, with their own layer-specific parameters (weight matrix $\boldsymbol{C}_i$ and bias vector $\boldsymbol{k}_i$ for a single-layer auxiliary network). In this scenario the activations for layer i at time step t become:

$$\boldsymbol{a}_t^i = \sigma(\boldsymbol{W}_i^\mathsf{T}\boldsymbol{x}_t + \boldsymbol{b}_i) + \phi_i(\boldsymbol{e}_t) \qquad \phi_i(x) = \sigma(\boldsymbol{C}_i^\mathsf{T}x + \boldsymbol{k}_i) \qquad 1 \leq i \leq L \tag{4.19}$$

[7]Concatenation to hidden layers is an alternative, e.g. as $\boldsymbol{x}_t^i = \begin{cases} \boldsymbol{x}_t & i = 0 \\ [\boldsymbol{a}_t^{i-1} \quad \boldsymbol{e}_t] & 1 \leq i \leq L \end{cases}$.

The hidden representations $\boldsymbol{a}_t^i$ are therefore decomposed into a SI term, $\sigma(\boldsymbol{W}_i^\top \boldsymbol{x}_t + \boldsymbol{b}_i)$, and a SD term, $\phi_i(\boldsymbol{e}_t)$, which is used to bias the SI hidden representations at training and test time. Alternatively, element-wise multiplication can replace the addition of SD and SI terms, to perform scaling instead of biasing as a form of normalisation. This normalisation is embedding-dependent, therefore the type and the reliability of the embedding has a great impact on the effectiveness and final accuracy of this SAT-DNN approach.

4.3.3 Embedding-based SAT-DNN

We explore different mapping functions $\phi(x)$ which learn to shift or scale the SI representations, to enhance their attribute invariance. The hypothesis is that with a more complex $\phi(x)$, the normalisation parameters can be learned to be more effective for normalisation. However, with more embedding-dependent parameters, the risk of overfitting to the embeddings grows. We evaluate how the capacity of the transformation of the attribute embedding influences the attribute-normalisation ability in the main part of the acoustic model (by using a control network, layer, vector, variable and constant scale to transform the auxiliary embedding), and we test whether the transformation of more abstract representations at the hidden layers is superior to the transformation of input features. We perform experiments with embeddings extracted at the utterance and speaker levels (extracted per frame), investigating the effectiveness of SAT-DNN with a limited quantity of adaptation data. Apart from various auxiliary networks $\phi(x)$, we also explore the effect of normalisation by shifting or scaling SI representations.

Control network

Several approaches have been proposed recently to incorporate an attribute embedding into the network to normalise the hidden representations. In Cui et al. [2017] i-vectors are mapped through a network to element-wise scaling and bias parameters. Our first approach to incorporate the embedding into acoustic model is similar to [Cui et al., 2017] and is depicted in Figure 4.6.

In our work, however, attribute-dependent mappings are generated by a control network from online i-vectors, not speaker i-vectors, to enable an efficient SAT-DNN. During decoding, the SAT-DNN model is adapted by extracting on-

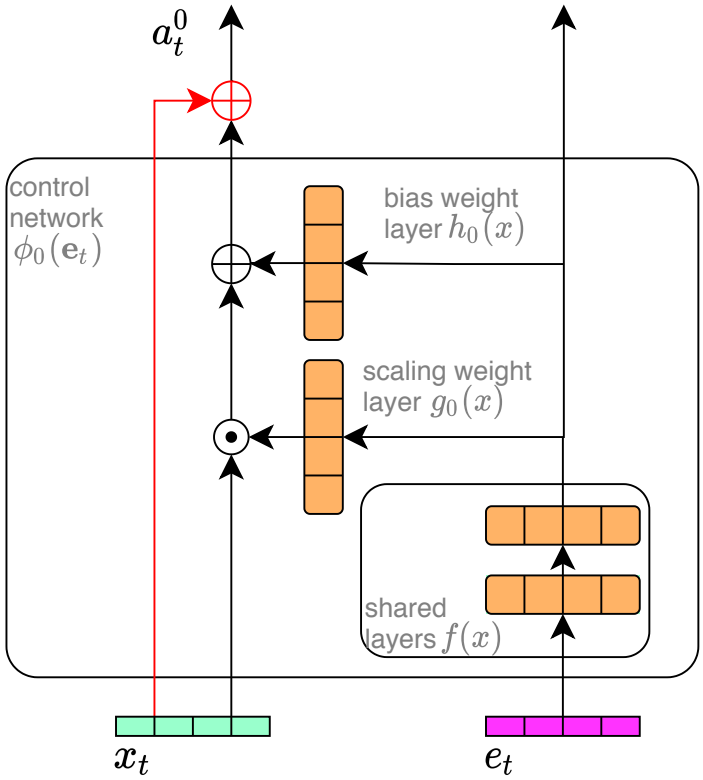


Figure 4.6: Embedding-based SAT-DNN using a control network with a skip connection (red) to learn the shift and scale to all hidden layers of the main part of the acoustic model. $\boldsymbol{x}_t$ is an input feature vector and $\boldsymbol{e}_t$ is an embedding at time step t.

line i-vectors for each test speaker at the frame level ($\boldsymbol{e}_t$), feeding the i-vector forward through the adaptation network $\phi(x)$, and integrating the generated shift and scale with the internal representations of the main part of the network. Differently to Cui et al. [2017], we also add a skip connection to the control network, in order to control the degree of adaptation at each layer. By doing this, instead of applying a control network just to one or two bottom layers of the main network, we may apply the control network to every SI layer i for L hidden layers in the network, without a loss in performance. At each time step t, the activations of the adaptively trained model can be computed as:

$$\boldsymbol{a}_t^i = \begin{cases} \boldsymbol{x}_t & i = 0 \\ \sigma(\boldsymbol{W}_i^\mathsf{T} \boldsymbol{a}_t^{i-1} + \boldsymbol{b}_i) + \phi_i(\boldsymbol{e}_t) & 1 \leq i \leq L \end{cases} \tag{4.20}$$

where $i = 0$ correspond to the DNN input and $\boldsymbol{x}_t$ is the input feature vector. σ is the $ReLU$ activation function. The shared layers of the control network, $f(x)$, use $ReLU$ non-linearity and are applied to an extracted speaker embedding (online

i-vector $\boldsymbol{e}_t$). Scaling ($g_i(x)$) and bias ($h_i(x)$) weight layers are used to scale and shift hidden representations, and are thus constrained to have the same number of units as the layers in the main part of the network. The scaling weight layer uses a sigmoid activation function to enforce positive scaling whilst the bias weight layer employs a hyperbolic tangent activation, enabling the bias to be positive or negative. Those layers act on the output of the shared layers and are applied to hidden representation to perform the normalisation (eq. 4.20). The output of the control network $\phi(x)$ depicted in Fig. 4.6 is defined as:

$$\phi_i(x) = \begin{cases} \boldsymbol{x}_t \odot g_i(f(x)) + h_i(f(x)) & i = 0 \\ \boldsymbol{a}_t^{i-1} \odot g_i(f(x)) + h_i(f(x)) & 1 \leq i \leq L \end{cases} \tag{4.21}$$

$$\begin{aligned} f(x) &= ReLU(\boldsymbol{B}^\mathsf{T}(ReLU(\boldsymbol{A}^\mathsf{T}x + \boldsymbol{k})) + \boldsymbol{m}) \\ g_i(x) &= sigmoid(\boldsymbol{C}_i^\mathsf{T}x + \boldsymbol{n}_i) \qquad\qquad 0 \leq i \leq L \\ h_i(x) &= tanh(\boldsymbol{D}_i^\mathsf{T}x + \boldsymbol{p}_i) \end{aligned}$$

where $\boldsymbol{A}$ and $\boldsymbol{B}$ are shared weight matrices, $\boldsymbol{k}$ and $\boldsymbol{m}$ are shared bias vectors, $\boldsymbol{C}$ and $\boldsymbol{D}$ are layer-specific weight matrices, $\boldsymbol{n}$ and $\boldsymbol{p}$ are layer-specific bias vectors.

We apply the control network at each layer i of the main network to learn the shift and scale to all of the hidden layers L. We also experiment with applying the control network only to the input features, as follows:

$$\boldsymbol{a}_t^i = \begin{cases} \boldsymbol{x}_t + \phi_i(\boldsymbol{e}_t) & i = 0 \\ \sigma(\boldsymbol{W}_i^\mathsf{T}\boldsymbol{a}_t^{i-1} + \boldsymbol{b}_i) & 1 \leq i \leq L \end{cases} \tag{4.22}$$

where $\boldsymbol{a}_t^i$ for $i = 0$ correspond to the normalised input feature space.

Control layer

Instead of using a multi-layer control network, we reduce the number of parameters acting on the embeddings by using a single control layer to transform the representations (Figure 4.7), to avoid overfitting. The transformation $\phi(x)$ becomes:

$$\phi_i(x) = \psi(\boldsymbol{A}_i^\mathsf{T}x + \boldsymbol{k}_i) \qquad 0 \leq i \leq L \tag{4.23}$$

This transformation is layer-specific, without any shared parameters across the network. In our experiments, we use the *ReLU*, *sigmoid*, *tanh* and *linear* activation functions ($\psi(x)$) to explore the effect of the linear vs. non-linear shift,

and the direction and value range of the shifts to the input features. $\boldsymbol{A}_i$ and $\boldsymbol{k}_i$ are the weight matrix and bias vector acting on the embeddings. With a control layer, we apply the shift to the input features $\boldsymbol{x}_t$ (as in Eq. 4.22), or all hidden representations (as in Eq. 4.20). The activations for scaling the representations are as follows, for input (Eq. 4.24) or hidden features normalisation (Eq. 4.25):

$$\boldsymbol{a}_t^i = \begin{cases} \boldsymbol{x}_t \odot \phi_0(\boldsymbol{e}_t) & i = 0 \\ \sigma(\boldsymbol{W}_i^\mathsf{T} \boldsymbol{a}_t^{i-1} + \boldsymbol{b}_i) & 1 \leq i \leq L \end{cases} \tag{4.24}$$

$$\boldsymbol{a}_t^i = \begin{cases} \boldsymbol{x}_t & i = 0 \\ \sigma(\boldsymbol{W}_i^\mathsf{T} \boldsymbol{a}_t^{i-1} + \boldsymbol{b}_i) \odot \phi_i(\boldsymbol{e}_t) & 1 \leq i \leq L \end{cases} \tag{4.25}$$

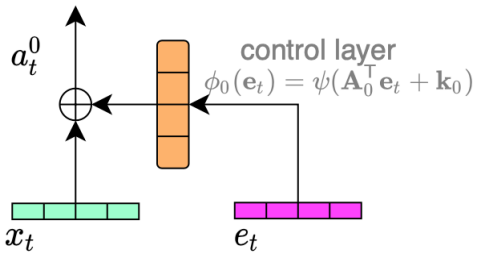


Figure 4.7: Illustration of a control layer acting on the embeddings e_t. Example for applying the shift to the input features $\boldsymbol{x}_t$.

Control vector

To further reduce the number of parameters applied to the embeddings, we estimate just the diagonal elements of the weight matrix $\boldsymbol{A}_i$, and we do not learn the layer-specific bias vector $\boldsymbol{k}_i$, such that the transformation $\phi(x)$ becomes:

$$\begin{aligned} &\phi_i(x) = \psi(\boldsymbol{A}_i^\mathsf{T} x) \quad 0 \leq i \leq L \\ &\boldsymbol{A}_i = diag(\boldsymbol{a}_i) \end{aligned} \tag{4.26}$$

The layer-dependent vector $\boldsymbol{a}_i$ is used to scale the embedding $\boldsymbol{e}_t$, and shift the input features – the activations can be derived from Eq. 4.22. Here, we use the sigmoid activation function for $\psi(x)$ to restrict the scaling of the embeddings to only positive values.

Control variable or constant scale

To further simplify the approach and reduce the number of parameters, a single layer-specific weight α_i can be used to scale the embedding prior to shifting the

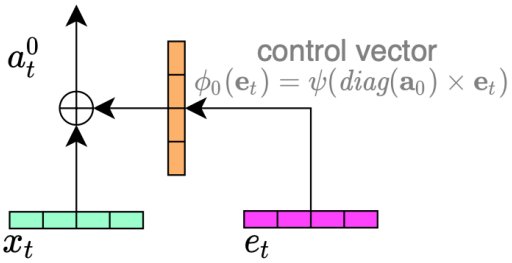


Figure 4.8: Illustration of a control vector acting on the embeddings e_t. Example for applying the shift to the input features $\boldsymbol{x}_t$.

input features (as in Eq. 4.22):

$$\phi_i(x) = \psi(\alpha_i \cdot x) \qquad 0 \leq i \leq L \tag{4.27}$$

where $\psi(x)$ is a *sigmoid* activation function. Moreover, to eliminate the need to learn any parameters acting on the embeddings, we also keep the scale α_i fixed (0.1) instead of learning the weight:

$$\phi_i(x) = \alpha_i \cdot x \qquad \alpha_i = 0.1 \qquad 0 \leq i \leq L \tag{4.28}$$

Finally, the concatenation of the input features and the embeddings can be seen as weighting the embedding with a constant scale α_i equal to 1. Then, $\phi_i(x) = x$, therefore, the activations become:

$$\boldsymbol{a}_t^i = \begin{cases} \boldsymbol{x}_t + \boldsymbol{e}_t & i = 0 \\ \sigma(\boldsymbol{W}_i^\mathsf{T} \boldsymbol{a}_t^{i-1} + \boldsymbol{b}_i) & 1 \leq i \leq L \end{cases} \tag{4.29}$$

with the weight matrix $\boldsymbol{W}_1$ acting on the input feature vector $\boldsymbol{x}_t$ and the embedding vector $\boldsymbol{e}_t$. Figure 4.9 shows the input features transformations by shifting with a control variable a_0.

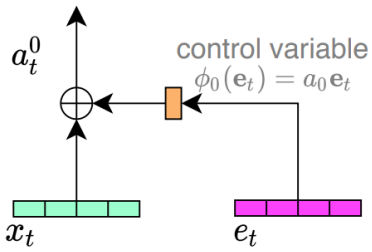


Figure 4.9: Illustration of a control variable acting on the embeddings e_t. Example for applying the shift to the input features $\boldsymbol{x}_t$. For a constant scale, $a_0 = const$.

The results for the proposed embedding-based SAT-DNN approaches are presented later in this chapter, in Table 4.8.

4.4 Experiments

We first provide the experimental results for the attribute-aware training, showing the influence of the type and quality of the embedding used for normalisation. We then explore real-time and low latency decoding constraints for the embedding extraction. Attribute-aware training experiments provide a reference for the methods developed for the adaptive training. We address the optimal structure for the auxiliary network, the normalisation approach, the effect of normalisation applied to all hidden layers vs. only at the input to the network, and the influence of the type of the embedding used for SAT-DNNs. A road-map for the experiments in this chapter is pictured below.

- Experiments in Chapter 4:
  - Section 4.4.1: **Attribute-aware training** for
    - Aurora-4
      - with TDNN
        - + offline i-vectors
        - + AM embeddings
    - MGB-3
      - with TDNN
        - + offline i-vectors
        - + AM embeddings
    - Financial and Political News
      - with DNN
        - + offline i-vectors
        - + online i-vectors
    - AMI IHM
      - with TDNN
        - + online i-vectors
      - with DNN
        - + online i-vectors
  - Section 4.4.2: **Auxiliary networks for adaptive training** for
    - AMI IHM
      - with DNN
        - + online i-vectors
          - and control network/layer/vector/variable/scale
            - used to scale/shift
            - input/hidden representations
        - + offline i-vectors
          - and control layer used to shift input features
        - + x-vectors
          - and control layer used to shift input features
        - + AM embeddings
          - and control layer used to shift input features

4.4.1 Attribute-aware training

We use four different corpora and different baseline models for attribute-aware experiments, therefore we describe each experimental setup separately. More detailed description of the corpora and the data split used in the experiments can be found in Chapter 3.

Aurora-4

We use Aurora-4 in our experiments to evaluate the attribute-aware training strategy for additive noise and mismatched microphone testing scenario, with

	WER				
Model	**A**	**B**	**C**	**D**	**Avg.**
TDNN baseline	3.64	7.69	8.96	19.45	12.53
+ *FBANK i-vectors (400)*	3.77	7.27	8.22	18.76	12.01
+ *MFCC i-vectors (400)*	3.71	**6.99**	7.86	**17.56**	**11.35**
+ *DNN embedding (400)*	4.38	8.22	12.15	20.12	13.33
+ *deep CNN embedding (400)*	4.54	8.02	9.37	20.05	13.02
+ *LDA (10) DNN embed.*	**3.58**	7.53	8.76	18.79	12.16
+ *LDA (10) deep CNN embed.*	3.65	7.37	**7.77**	17.99	11.68

Table 4.2: WER (%) for Aurora-4 test sets.

multi-condition training. Aurora-4 training set is also rather small – approx. 15h – therefore we were able to perform the initial experiments relatively quickly.

We train the TDNN baseline for Aurora-4, and we examine different types of auxiliary features to inform the network about the attributes of the utterances. The TDNN baseline is trained on raw 13-dimensional MFCC features, without mean and variance normalisation – a role of the embeddings is to normalise the input feature space[8]. The baseline comprises 5 TDNN layers, each with 650 units. It has a left context of 13 and right context of 7. All Aurora-4 models in this section use the alignments generated by a triphone GMM model. The WERs for Aurora-4 TDNN baseline as well as for models using different embeddings to inform the network about the utterance attributes are presented in Table 4.2.

i-vectors for Aurora-4 and MGB-3 tasks in this section have 400 dimensions and are extracted using MFCC or FBANK input features. The i-vectors are offline, extracted per utterance for both training and test sets. This way of extraction is also used for DNN and deep CNN embeddings. An LDA-like transformation layer[9] is used in all TDNN experiments, to scale down "non-informative" dimensions, after the concatenation of the i-vectors with spliced input features (with 2 frames of right and left context). This fixed affine transform has the effect of encouraging stochastic gradient descent to ignore non-informative values. It is learned in a supervised way prior to learning the parameters of the following

[8]We provide the results for CMN normalised MFCCs with auxiliary features for AMI IHM in Section 4.4.1 and find only a slight (0.1% relative) difference in WERs, compared to unnormalised MFCCs with auxiliary features.

[9]Described in Appendix C.6 in [Povey et al., 2015].

layers, using the CD states for supervision.

In noisy conditions (test sets B and D), i-vectors computed on MFCC features perform best. Providing a representation which is more noise-aware but less speaker-aware[10] (DNN and deep CNN embeddings) degrades the ability of the acoustic model to recognise tied-state triphones.

We can recover some of the model's performance by applying an LDA transform to the embeddings. An LDA transform projects the embeddings into a lower-dimensional space with good between-speaker separability, hence the DNN and deep CNN embeddings transformed with LDA are more speaker-aware and less noise-aware. They are then better suited for acoustic model adaptation. For clean test sets (A and C) LDA transformed NN embeddings outperform i-vectors, with much smaller dimensionality (10 dimensions).

MGB-3

We use the MGB-3 corpus to investigate if the findings for the Aurora-4 corpus are valid for a larger training set (approx. 350 h). It is worth noting that the MGB-3 corpus is much more challenging than Aurora-4, especially due to imperfect transcriptions for the training set, overlapping speech, and various types of background noise. We evaluate the attribute-aware training strategy with various embeddings in this real-life ASR scenario.

We use the same types of embeddings as auxiliary features for the MGB-3 task. The TDNN baseline for MGB-3 is a 5 layer TDNN model with 1280 units in each layer. The input features are 40-dimension high resolution MFCCs, appended with 3 pitch features. The model's left and right context is 11 and 9 frames respectively. All MGB-3 TDNN models use the same alignments, generated from a sequence discriminative trained TDNN model with i-vector adaptation. The results for the MGB-3 dataset are in Table 4.3. 100-dimension i-vectors extracted on FBANK features were the representations providing the most gains over the baseline TDNN model. Similar to the results for Aurora-4, using raw (PCA) DNN or deep CNN embeddings as auxiliary features resulted in a performance degradation (WER worse than unadapted baseline or close to it). However, we found that training for two epochs instead of four resulted in a lower WER, indicating faster and more effective training with the embeddings, compared to the baseline.

[10]The information encoded in the embeddings is analysed in Chapter 6.

Model	WER
TDNN baseline	29.4
+ *FBANK i-vectors (400)*	27.9
+ *FBANK i-vectors (100)*	**27.2**
+ *MFCC i-vectors (400)*	27.7
+ *MFCC i-vectors (100)*	27.5
+ *DNN embed. (400)*	29.8
+ *DNN embed. (400, 2e)*	28.6
+ *deep CNN embed. (400)*	29.3
+ *deep CNN embed. (400, 2e)*	28.6
+ *LDA (10) DNN embed.*	28.8
+ *LDA (10) deep CNN embed.*	28.7
+ *LDA (70) deep CNN embed.*	**27.7**
+ *LDA (70) deep CNN embed. (2e)*	28.4

Table 4.3: WER (%) for MGB-3 English dev17a test set. 2e means training for 2 epochs.

We also used the LDA transform[11] to improve the speaker-awareness of the embeddings. This strategy proved to be effective. The WER for all segments for the LDA transformed deep CNN embeddings match the WER for 400-dimensional MFCC i-vectors but with much lower dimensionality (70).

We have shown that the embeddings extracted in a non-causal framework (offline) at the utterance level are effective for AM adaptation in the attribute-aware strategy – up to 13% and 7% relative WER reduction for Aurora-4 and MGB-3, respectively. The performance of the embeddings extracted from neural networks was similar to the performance of the i-vectors.

Financial and Political News

Next, we explore attribute-aware training for financial and political news domain, where the speaker changes are frequent. This is a similar domain to MGB-3, but with more training data – approx. 830 hours. Here, we explore the embeddings extracted at the frame level (for use in online decoding) and compare the per-

[11] Trained on MGB-3 training set labels corresponding to the colours of the captions, indicating speaker changes rather than the actual speakers.

Training i-vectors	Testing i-vectors			
	utterance	speaker	spk cluster	session
spk cluster	24.3	21.6	21.9	24.5
utterance	23.8	**21.4**	21.7	24.6

Table 4.4: Baseline WER results for Financial and Political News data. Offline i-vectors with the sufficient statistics accumulated across utterances, speakers, speaker clusters, or sessions.

formance with utterance level ones. We explore different boundaries (session, speaker clusters, true speakers, utterances) used for the embedding extraction, as well as causal (online) and non-causal (offline) extraction frameworks, for training and testing. The goal is to find an optimal attribute-aware setting suitable for online decoding, and to compare its performance with offline decoding.

We start by training attribute-aware baseline models for offline decoding, with offline i-vectors used in training and testing. The i-vectors differ by how the sufficient statistics are accumulated. The embedding extraction and attribute-aware training is more challenging when the speaker change occurs frequently. For training, speaker clustering can be performed automatically, to infer speaker clusters boundaries and extract the embeddings within speaker clusters. i-vectors can therefore be extracted with the sufficient statistics accumulated across utterances or speaker clusters. For the experiments, we could also use gold speaker labels (for test set), so we also evaluate i-vectors with statistics accumulated across speakers. The baseline results for the offline setting, with offline i-vectors used for both training and test sets are in Table 4.4.

The acoustic models in Table 4.4 are 6-layer DNNs with 2048 units in each layer, resulting in $\sim$35 million parameters. The input features (at the network input and for i-vector extraction) are fMLLR transformed MFCCs and the WERs for the models without i-vector adaptation are 25.4%, 22.4%, 22.5%, 24.3% for utterance, speaker, speaker cluster, and session input features normalisation for the test sets. The adaptation with offline i-vectors can therefore bring the improvements over the fMLLR baseline.

In general, for offline decoding, training with utterance i-vectors is superior to training with speaker cluster i-vectors. This result can be explained by more variability introduced by the segmental i-vectors at training, compared to using

SS accumulation	Testing i-vectors		
	offline	online (frame-wise)	online (segmental)
speaker	22.1	22.1	22.0
spk cluster	22.3	21.9	21.7
utterance	22.2	22.8	–

Table 4.5: Baseline WER results for Financial and Political News data, for different i-vector types, with the sufficient statistics accumulated across utterances, speakers and speaker clusters.

the same i-vector for every utterance within a speaker cluster. More diverse auxiliary embeddings might diminish the risk of overfitting, and contribute to better final performance.

For the test set, on the other hand, more context (speaker and speaker cluster) is superior to utterance i-vectors. The final model performance is dependent on the quality of the i-vectors. The lowest WER was achieved for the most reliable i-vectors, extracted with the use of true speaker labels. However, the utterance i-vectors might be preferred for lower latency during decoding.

Next, we address real-time or low latency decoding scenarios with the i-vectors extracted in an online fashion. As previously discussed, online i-vectors are causal – there is an update of the sufficient statistics, taking into account past frames or utterances, within utterances or speakers (see Figure 4.4). For *real-time* decoding, the i-vectors for the test set need to be extracted at the frame level, and for *low latency* decoding at the utterance level.

We train a smaller (∼9 million parameters) DNN baseline SI model on top of high resolution MFCCs (40D), without input feature normalisation – a smaller model might be preferred for real-time applications. Training i-vectors were extracted online, per frame (every 10th frame), carrying the speaker cluster information. Also, to increase the variability of the embeddings used for training, a constraint of two utterances per speaker cluster was applied – the speaker clusters obtained at the preprocessing step were further divided before i-vector extraction. The WER for the unadapted baseline was 23.6%. The summary of the results for SA models with different i-vectors used for the test set is presented in Table 4.5.

With online i-vectors used for training and offline test i-vectors, the clustering of the test data still brings the improvement over the SI baseline (22.3%

over 23.6%), but a similar performance can be achieved with simply using the utterance-level embeddings (22.2%). Therefore, when the speaker information is not available, segmental offline i-vectors should be preferred. There is not a big difference in WERs compared to speaker offline i-vectors (22.1%).

More improvements can be obtained by using online test i-vectors, especially when the statistics are accumulated across speaker clusters (21.9% and 21.7%, for frame-wise and segmental i-vectors, respectively). Because of the causal nature of the online i-vectors, and the exponential decay used to bias the statistics towards the most recent history, the model learns to normalise the representations more effectively. Overall, the best setup for real-time decoding is to accumulate the statistics across speaker clusters.

It is interesting that the speaker cluster information used for the i-vector extraction performs better than the gold speaker labels. This can be explained by the type of labels used for the training i-vectors (speaker clusters instead of gold speaker labels), and therefore matched i-vectors in training and testing. Another possible explanation is that speaker clusters may contain not only the utterances from the same speakers, but may also capture other acoustic characteristics of the environment, enabling normalisation of multiple recording attributes.

For real-time decoding, the best approach is to use the speaker clustering information to extract online frame-wise i-vectors (21.9%). This approach requires however the real-time clustering (or speaker boundaries) information to be available, which is non-trivial. In our experiments, we used the clusters obtained from pre-processing – available beforehand. For true real-time decoding approach, utterance level frame-wise online i-vectors can be used, with 0.8% absolute WER reduction over the SI baseline. When low-latency decoding is acceptable, online segmental i-vectors accumulated across speaker clusters are optimal.

For offline decoding and offline i-vector extraction, the most optimal setup was to use utterance-level i-vectors for training and speaker-level i-vectors for testing. For real-time decoding and online i-vector extraction, the most optimal setup was to use online frame-wise i-vectors, with statistics accumulated across speaker clusters, for both training and testing. Those results indicate that introducing variability to the training embeddings is beneficial, and that matched training and testing i-vector extraction (online frame-wise across speaker cluster) could lead to optimal performance. Using speaker clusters instead of true speakers labels was superior in our experiments, however, an additional experiment with

Model	WER
Kaldi TDNN (sp)	26.9%
Kaldi TDNN + CMN (sp)	26.6%
Kaldi TDNN + online i-vectors (sp)	26.2%
Kaldi TDNN + online i-vectors + CMN (sp)	26.1%

Table 4.6: Kaldi baseline results for AMI IHM eval set. *sp* - speed perturbation applied to the training set for data augmentation. + online i-vectors means concatenation to the input, + CMN means mean normalisation of the input features.

true speaker labels used in training and testing could determine if the gains were due to matched training and testing i-vector extraction, or different acoustic characteristics captured by the clusters, enabling for more effective normalisation.

AMI IHM

We also evaluate the attribute-aware training on the AMI corpus. AMI is a corpus of meeting recordings, and with IHM training condition, the acoustics are not subjected to additive noise as in Aurora-4 or Financial and Political News data. Therefore, the attribute-aware training for AMI IHM centres on speaker normalisation. The attribute-aware training experiments for AMI IHM serve as a reference for DNN-SAT with auxiliary networks experiments.

We use the train/dev/eval set split defined by the AMI Kaldi recipe[12]. WERs reported in Table 4.6 and 4.7 are for the AMI IHM eval set. The AMI IHM train set was used for training, AMI IHM dev was used to evaluate the models after each epoch. Input features were high resolution MFCCs (40 mel bins and 40 coefficients) with delta and delta-delta features with CMN. The left and right time frame context was 5. We use online i-vectors as the auxiliary embeddings, which are extracted at every time frame by accumulating the statistics within speakers for up to 30 seconds for training. At test time, utterance-level online i-vectors are used. The TDNN baseline has 5 layers with 650 units.

Table 4.6 confirms that using online i-vectors can be an effective feature augmentation technique (26.2%). The gain is bigger than normalising the input features with CMN (26.6%), and combining CMN with online i-vectors augmentation is slightly more beneficial (26.1%). However, default Kaldi recipes that

[12] egs/ami/s5b/local/chain/run_tdnn.sh

Model	WER
Pytorch DNN + CMN	27.0%
Pytorch DNN + online i-vectors + CMN	26.5%
Pytorch DNN + scaled online i-vectors (fixed scale) + CMN	26.1%
Pytorch DNN + scaled online i-vectors (trainable scale) + CMN	26.1%

Table 4.7: Pytorch baseline results for AMI IHM eval set (without data augmentation). + online i-vectors means concatenation to the input, + CMN means cepstral mean normalisation of the input features.

incorporate i-vectors into the acoustic model, make the use of an additional fixed LDA-like transform applied to the conactenated features, which is scaling down the dimensions that are “non-informative". We implement the attribute-aware training setup in Pytorch to disentangle the gains from the information provided by the embeddings from the gains from additional Kaldi-specific transformations.

DNN baseline models trained with Pytorch confirm the importance of adjusting the values of the auxiliary embeddings. Appending i-vectors to the input in PyTorch brings the improvement, however, we find that manually scaling down the whole i-vector representation (scale with a constant = 0.1) brings more improvements. This result is a motivation to search for an optimal transformation of the embeddings in an adaptive training strategy.

4.4.2 Auxiliary networks for adaptive training

In this section, we use the AMI IHM dataset for speaker-adaptive training experiments, with the same train/dev/eval split as in Section 4.4.1. Although the AM embeddings which are used for normalisation can in principle represent multiple attributes, in this section they are extracted from the acoustic model trained on AMI IHM data. This dataset does not include different factors of acoustic variation, e.g. different noise types added artificially, therefore the AM embeddings are expected to be more speaker-representative. Hence, with all of the embeddings in this section, we aim to normalise the factors associated with speaker variability.

We use the Kaldi toolkit [Povey et al., 2011] for input acoustic feature extraction (40-dim high-resolution MFCC features with double deltas and 5 frames of context at each side, with cepstral mean normalisation (CMN) applied except where specified), i-vector and x-vector extraction, training initial models for align-

ment, and decoding. We use the PyTorch-Kaldi [Ravanelli et al., 2019] toolkit to implement DNN acoustic models and Tensorflow [Abadi et al., 2015] to extract deep CNN embeddings. The acoustic model is a 6 layer DNN with 2048 units in each layer, trained with cross-entropy loss over 3984 context-dependent tied triphone states.

4.4.2.1 Types of embeddings

In this section, the i-vectors are extracted from MFCC features. For training we use online speaker i-vectors, with two utterances per speaker. For test i-vectors we extract either online frame-wise representations with carrying the statistics across speakers for real-time decoding, or offline per utterance i-vectors for low-latency decoding[13]. We explore both types of i-vectors for efficient SAT-DNN. i-vectors are designed to capture both speaker and channel characteristics, as they use a single variability subspace to model different types of variability in the speech signal.

x-vectors are extracted per utterance; however, since the x-vector extractor is trained with the use of speaker labels, they are explicitly designed to capture speaker characteristics. Compared to i-vectors, x-vectors should therefore be invariant to within-speaker channel variability. We used a pre-trained SRE16 x-vector model[14] to extract the x-vectors.

Deep CNN embeddings are also extracted per utterance. Here, the speaker labels are not used in the embedding extraction. The model used to extract the embeddings is a very deep CNN acoustic model with 2D 3x3 kernels, trained to classify senone states [Rownicka et al., 2017]. PCA is used to reduce the dimensionality of the embeddings. To improve speaker discrimination to the embeddings, an LDA transform informed by training speaker labels may be used. We show in Chapter 6 that PCA CNN embeddings are more characteristic of the acoustic condition than i-vectors (for Aurora-4 dataset), and LDA CNN embeddings are also better speaker representations, compared to i-vectors.

Embed. mapping	shift/scale	WER
SI baseline	-	28.3
CMN	-	27.0
control network	$\oplus$, $\odot$	27.0
control layer	$\odot$	26.5
control layer	$\oplus$	**25.9**
control vector	$\oplus$	26.1
control variable	$\oplus$	26.1
control scale	$\oplus$	26.1
embedding concatenation		26.5
control network (hidden)	$\oplus$, $\odot$	26.1
control network (hidden)	$\oplus$	26.0
control network (hidden)	$\odot$	26.0
control layer (hidden)	$\oplus$	26.4
control layer (hidden)	$\odot$	27.4

Table 4.8: Comparison of the approaches to generate the parameters acting on the embeddings for input features and hidden representations transformation for AMI IHM eval. All models (except for the SI baseline) use CMN features. Element-wise addition is denoted with $\oplus$ (shifting), and $\odot$ denotes element-wise product (scaling).

4.4.2.2 Embedding incorporation

We show empirical results for all mentioned approaches to map the online speaker i-vectors to control parameters used to transform input features as well as hidden representations.

The results for the different embedding mapping and transformation approaches are presented in Table 4.8. We find that the embedding-based adaptive training yields lower WERs than embedding-based attribute-aware training (i.e. concatenation of the embedding to the input features), hence the embedding incorporation can be optimised by using embedding-specific parameters. The capacity of the transformation acting on the embedding (network, layer, vector, variable), the type of normalisation (shift or scale), as well as the position in the network (input or hidden normalisation), determines the final adaptation

[13] Also, to match the extraction framework of the deep CNN embeddings for fair comparison
[14] http://kaldi-asr.org/models/m3

performance.

The first block gives baseline results, the second one is for input feature transformation, and the third is for hidden representation transformations. Different patterns can be observed depending on where in the network the transformation is applied, however, almost all SAT-DNN strategies outperform the attribute-aware training (*embed. concat.* in the table). For input feature transformation, a multi-layer control network did not outperform the CMN baseline. All of the approaches with fewer parameters were superior to the control network.

For the control layer, shifting the input features was more important than scaling them with the activations generated from the embeddings. The input features are already mean-normalised per speaker, so the transformations are potentially better suited to normalise different factors of variation in the utterances. This could explain better performance of shifting rather than scaling with the control layer – noise might be compensated by shifting.

For the control layer acting on the embeddings, we experimented with different activation functions, finding that all of them give similar performance. The simplest and at the same time the most flexible approach is to use a linear activation function. It does not restrict the direction and the value range of the feature shifts. When using a linear identity activation function we do not benefit from learning more abstract embedding representations prior to the input feature transformation – so the advantage of this approach lies in scaling the embeddings with the weights learned in the control layer.

Learning the control variable did not bring gains over a fixed control scale, however, we evaluated how it changed over training (Figure 4.10). It is interesting that using a larger scale for the i-vector representation as the training progresses gives good performance. This shows that it is especially important to scale down the embedding at the beginning of the training to prevent the model from overfitting; scaling down as the training progresses becomes less important.

Furthermore, we found the training strategy for training with the auxiliary networks acting on the embeddings to be crucial. The most effective approach was to initialise the main part of the network with the parameters from the SI baseline in the first stage, and in the second stage to train the remaining control parameters, together with fine-tuning the parameters of the main part of the network. This approach was much better than fixing the parameters of the main part of the network in the final stage and only training the control part of the

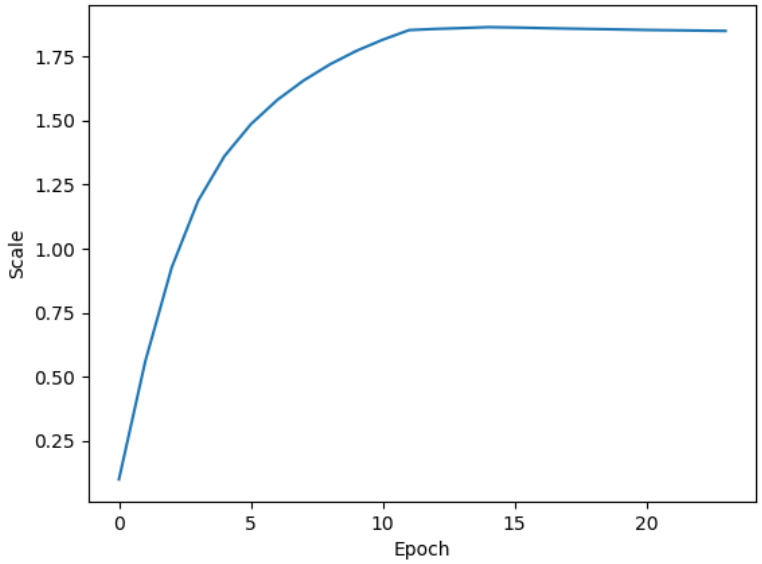


Figure 4.10: Evolution of control variable during training for AMI IHM (before *sigmoid* activation function).

network. We hypothesise that this is due to the ability to update the parameters of the DNN model in the new, normalised feature space. The training strategy is the factor differentiating the embedding concatenation vs. other approaches. It is therefore important to scale the embedding, as well as to use a multi-step training approach to generate the control parameters for the input feature transformation in the SAT-DNN scheme.

The results for the transformations applied to the hidden layers (third block in Table 4.8) show that, overall, transforming hidden representations is not more effective than transforming the input features. This might be explained by the fact that batch normalisation is used in our experiments, so the hidden representations might be already invariant to different speech attributes. Further transformation with control parameters derived from the embeddings is not as effective as transforming the input features, allowing the main part of the network to operate in the normalised feature space at the first hidden layer. Interestingly, the hidden layers benefit from a different mapping approach to the input layers: in this case, the control network with the most flexibility to generate transformation parameters is superior to other approaches. We conclude that learning a flexible layer-wise mapping is important for hidden layer transformations.

4.4.2.3 Embedding type influence

The results presented so far were obtained for online i-vectors as the auxiliary network input. We evaluate the influence of the type of embedding on the SAT-DNN task, using a control layer to shift the input features (the optimal online i-vector incorporation approach). The results are presented in Table 4.9. By replacing online speaker i-vectors with offline utterance i-vectors, we gained 0.1% in WER, thus the frame level variation of the embeddings did not contribute to better WER in our experiments. On the other hand, online decoding can only be performed with online representations, and as the loss of performance is not substantial, one may prefer to use online i-vectors. It is important to bear in mind however, that genuine speaker labels were used for online i-vector extraction in our experiments. For the deployment of SAT-DNN with online representations, an additional diarisation or speaker clustering step would be needed.

The PCA transform in the deep CNN embedding extraction framework is used simply to reduce the dimensionality of the embedding in an unsupervised fashion. Since speaker information is not used in the embedding extraction, PCA CNN embeddings can be regarded as utterance summaries, capturing local acoustic characteristics. Their incorporation into the acoustic model gives a substantial improvement over the baselines; however, adding speaker discrimination to the utterance summary (using LDA) further improves the results.

Interestingly, better speaker discriminability does not guarantee better normalisation in SAT-DNN. 20D embeddings were the most discriminative for speaker recognition (see Chapter 6), and yet 100D embeddings were optimal for SAT-DNN. It is possible that by using more dimensions for the embeddings, we are able to capture the utterance-level representation beyond speaker identity, resulting in better SAT-DNN performance. x-vectors are extracted from a network trained to classify speakers, hence they are more invariant to within-speaker variability, caused by speech attributes other than speaker identity. We believe that this might the reason for poorer ASR performance with SAT-DNN.

The differences in WERs stemming from the use of different embeddings in SAT-DNNs motivates our work presented in Chapter 6, where we analyse the information encoded in the embeddings, to link their adaptation performance with their extraction framework.

Embedding type	WER
SI baseline	28.3
CMN	27.0
online speaker i-vectors	25.9
offline utterance i-vectors	**25.8**
CNN embeddings (100D PCA)	26.2
CNN embeddings (100D LDA)	**25.8**
CNN embeddings (20D LDA)	26.1
CNN embeddings (100D PCA + 100D LDA)	25.9
x-vectors (pretrained, AMI backend, 100D LDA)	26.3
x-vectors (pretrained, AMI backend, 40D LDA)	26.4

Table 4.9: WERs for SAT-DNN with different embeddings. Models are trained with a control layer used to shift the input features.

4.5 Conclusions

In this chapter, we used the embeddings to disentangle learned representations and to improve the robustness of the acoustic models. We showed that training a deep neural network acoustic model with an embedding is an effective adaptation method in various speech recognition tasks. Using auxiliary networks acting on the embeddings provided an optimal deep representation in our experiments. The proposed methods enabled fast and unsupervised adaptation, without using a disjoint adaptation dataset.

In attribute-aware training, without embedding optimisation, we were able to reduce the WER with various types of embedding. We found that by improving speaker-discriminability of the embeddings with an LDA transform, trained with the supervision of speaker labels, we improved the performance of the embeddings for the acoustic model adaptation. For Aurora-4, LDA AM embeddings outperformed i-vectors for test sets without additive noise. For MGB-3 English, they did not outperform the best performing i-vectors, but they matched the performance of the 400-dimensional i-vectors extracted on top of the MFCC features. One possible explanation for this result is that the LDA transform for MGB-3 was not trained reliably, due to the colours of the captions used as speaker labels, instead of the actual speaker labels. In the absence of speaker meta data, this approach can be effective, but can also be improved with more reliable speaker

information. Therefore, the data used for LDA transform extraction is crucial for optimal AM embeddings performance in the attribute-aware training. Experiments on Financial and Political News data have shown that for the best accuracy for low-latency decoding, speaker cluster information can be used to extract the embeddings in an online fashion, with per utterance sufficient statistics update and i-vector extraction. For real-time decoding, frame-wise i-vectors can be used with small performance degradation.

We also investigated the embeddings for adaptive training, using the auxiliary networks to optimise the embeddings. We confirmed the superior performance of SAT-DNN over attribute-aware training, with 25.8% for the best SAT-DNN model over 26.5% for the concatenation of the embedding to the input features. Matched-pairs significance test finds a significant difference between those systems at the level of $p = 0.001$. We evaluated the influence of the flexibility of the mapping applied to the embeddings, as well as transforming input features compared to hidden representations. We found that transforming hidden layers is not more effective than learning shifts to the input features. With this approach, and an appropriate training strategy, the main part of the network is updated in the normalised feature space. Although using a multi-layer control network to normalise all hidden representations gave similar performance, the simpler approach of linearly shifting the input features with single layer activations learned from the embeddings should be preferred. Only differences in WERs between the systems of 0.2% absolute and over are statistically significant.

We evaluated the effect of the embedding type on SAT-DNN ASR performance. We found that frame-level variation of the embeddings did not bring WER improvements in embedding-based adaptive training – utterance-level summaries were the most beneficial for SAT-DNN. Utterance summaries, such as the proposed AM embeddings, have the potential to capture more speech attributes than just the speaker identity. Adding speaker discrimination to the utterance summary was useful, but we found that the best speaker discriminability of the embeddings did not correlate with the best performance in SAT-DNN. In fact, it might be more important for the SAT-DNN embeddings to capture additional utterance attributes, rather than focusing solely on speaker differentiation. This can be achieved by i-vectors and deep CNN embeddings, but perhaps not by x-vectors in the current extraction framework. Adding channel or acoustic condition discriminability to x-vectors, e.g. by using multi-task learning for the x-vector

extractor, could improve their performance for SAT-DNN.

We showed that the incorporation of the embedding to the SI acoustic model can be optimised, and that the type of the embedding used in this SAT-DNN scheme influences the adaptation performance. Those findings motivated us to further analyse utterance embeddings, to better understand the connection between their extraction framework and their AM adaptation performance – we address this aspect in Chapter 6.

CHAPTER 5

Multi-scale representations for robust and efficient modelling

5.1 Overview

This chapter is based on [Rownicka et al., 2020] and proposes a multi-scale octave convolution layer to learn robust speech representations efficiently.

Similarly as for the embedding-based adaptation, with multi-scale representation learning we aim to disentangle the underlying factors of variation changing at different rates to learn more robust acoustic representations. We also consider computational and memory efficiency. The proposed method enables to quickly adapt to testing conditions in an unsupervised manner, since a disjoint adaptation dataset is not required for the adaptation.

Our main research question is:

- Can multi-scale octave convolution provide more robust representations than vanilla convolution?

Additionally, we explore the following:

- How does the input representation type influence multi-scale representation learning performance?

- Can multi-scale learning be more effective with augmented training sets?

In Section 5.2, we briefly review the work related to learning multi-scale representations for ASR, and to efficient CNNs. We then describe the proposed methods in Section 5.3 and we present and analyse the results in Section 5.4. The conclusions can be found in Section 5.5.

5.2 Background

This section provides a brief literature overview of multi-scale representations in ASR. We also discuss related work on efficiency in CNN-based architectures.

5.2.1 Multi-scale representations

Multi-scale processing has been previously proposed for a variety of speech recognition tasks. Wavelets are a prime example of multi-resolution processing functions – they are used to divide the data into different frequency components, and to analyse each component with a resolution matched to its scale. For speech recognition, where the frequency is the most important factor for distinguishing phones, a basis which carries both temporal location and frequency content can be an advantageous modelling approach.

The Fourier transform is commonly used to obtain the frequency information, however, it is only localised in frequency. The Fourier transform can be thought of in the terms of the Heisenberg Uncertainty Principle [Heisenberg, 1927], which states that two quantities (here, frequency and temporal spread) cannot be made arbitrarily small simultaneously[1]. Gabor [1946] first applied this principle to signal processing. Since speech is a transient signal – localised in time and frequency – wavelets, which have a multi-scale behaviour, have been applied to speech processing over the years. For example, Gupta and Gilbert [2001] use wavelet coefficient features for robust speech recognition. Similarly, Kotnik et al. [2003] use wavelet packet decomposition for feature extraction. Choueiter and Glass [2007] extend wavelet-based frameworks for phonetic classification. They design new wavelets using filter design methods, and improve the flexibility in frequency partitioning by implementing rational as well as dyadic filter banks.

[1]The short-time Fourier transform (STFT) addresses those issues and it is commonly used in speech applications. See sec. 2.1.2 for more information.

The gains over traditional wavelets as well as the MFCC baseline are presented for the TIMIT task. Adiga et al. [2013] explore gamma-tone wavelet for more robust ASR. Wavelet and multi-resolution auditory models were also previously explored by Yang et al. [1992]; Irino and Patterson [2002]. A review of many other application of wavelets in speech processing (such as speech detection, separation, enhancement, noise suppression, speaker identification, emotion recognition, coding, synthesis, compression, quality assessment, pathology and forensic analysis) can be found in [Farouk, 2013].

Deep Scattering Spectrum (DSS) was introduced by Mallat [2012]; Anden and Mallat [2014] to represent different time scale information of a raw acoustic signal in the form of wavelet filters. The general tree structure of the transforms, as well as local translation invariance make the approach analogous to deep convolutional networks. However, differently than in CNNs, DSS features are not learned – they are set in order to achieve pre-defined invariances. The scattering transform can also be seen as a generalisation of FBANKs; it keeps all of the information about the input at each level, preserving higher resolution information (unlike CNNs), and automatically yields a sparse high-dimensional representation. Peddinti et al. [2014] explored how to effectively use DSS features with CNN acoustic models. They addressed normalisation, regularisation, and model topology approaches. A relative improvement of 7% was reported for DSS features with a CNN for TIMIT, compared to FBANK features.

An example of more recent work relying on the multi-resolution processing concept for speech recognition is presented in [Tòth, 2017]. Here, a multi-resolution spectrum is used as input to a CNN model – gradually coarser resolution is used for more distant frames, enabling for the enlarged receptive field. Chen et al. [2018] propose Big-Little Net (bL-Net), a multi-scale feature representation for visual and speech recognition. This network also uses a CNN architecture and focuses on the efficiency. The key idea is to use a high-complexity branch (accurate but costly) for low-scale feature representation and a low-complexity branch (efficient but less accurate) for high-scale feature representation. bL-Net is designed as a replacement block for ResNet [He et al., 2016] – a model with residual connections. The model proved to be effective, with improved accuracy and reduced computations for image tasks. The improvements in terms of the accuracy are smaller for the speech recognition task, however, the proposed method still provides an efficient and effective way to model speech representations. von

Platen et al. [2019] propose a raw waveform based multi-span structure for a CNN acoustic model. Multiple streams of CNN input layers are used, each processing a different span of the raw waveform signal. Evaluation on CHiME-4 and AMI show improved results (5% relative) over the FBANK baseline.

5.2.2 Efficient CNNs

Deep convolutional neural networks (CNNs) with 2D convolutions and small kernels [Simonyan and Zisserman, 2015], have achieved state-of-the-art results for several speech recognition tasks [Sercu et al., 2016b; Sercu and Goel, 2016b; Yu et al., 2016b; Qian and Woodland, 2016; Tan et al., 2018]. As discussed in Chapter 2, deep CNN models do not have a great amount of parameters compared to fully-connected models, because the weights are shared within a feature map. However, the number of connections can be very high. The number of connections is in turn correlated with the number of computations.

The accuracy of deep CNN models grows with their complexity. In spite of improved accuracy, learned representations can be redundant. Several approaches have been proposed in the literature to reduce this redundancy, and therefore to improve their efficiency. Neural Network pruning consists in removing redundant parameters from the network. It is not a new concept [LeCun et al., 1989b], but it has recently gained more popularity. Resulting lower computational cost and memory footprint are desired features for real-time decoding and deployment on mobile or embedded devices.

One possible approach to address the redundancy is to rank the neurons in order to remove the low-ranked ones and obtain smaller and faster models. Recently, Han et al. [2015] have shown that magnitude-based pruning of weights is an effective approach for the VGGNet. However, weight pruning in the convolutional layers introduces the need for sparse BLAS libraries or specialised hardware [Han et al., 2016]. Also, sparse data structures create additional storage overhead. Moreover, most of the removed parameters of the VGGNet come from the fully connected layers, where the computational cost is low, so overall, the computation time is not much reduced.

Li et al. [2016] address those problems by applying structured pruning in the convolutional models, i.e. they remove the whole filters together with their connecting feature maps. This approach introduces structured sparsity instead

of sparse connectivity patterns. Polyak and Wolf [2015] also address channel-wise redundancy for run-time acceleration in a face identification task, which is considered to be less redundant, making pruning more challenging. Huang et al. [2017] propose CondenseNet to reduce computations with learned group convolutions. Here, the focus is also channel redundancy, but the approach is to reduce computations directly during training, not pruning and fine-tuning a pre-trained model as in previous works. Similarly, Luo et al. [2019] propose ThiNet to address channel redundancy and aim for acceleration and compression for mobile devices in training and testing. Here, however, whether a filter can be pruned depends on the statistics of the next layer, not its own layer, which differentiates ThiNet from previous methods. The authors show state-of-the-art results in several computer vision tasks (e.g., classification, detection, segmentation). They also propose "gcos" (Group COnvolution with Shuffling) – a more accurate group convolution scheme, to further reduce the pruned model size. Ma et al. [2018] proposed previously a similar idea with ShuffleNet, however, the more recent CondenseNet outperforms ShuffleNet in terms of computational efficiency at the same accuracy level.

Tan and Le [2019] use a compound coefficient (i.e. a constant scaling coefficient for width, depth and resolution of the network) to scale up CNNs in a more structured manner. Neural architecture search is used to obtain a family of models, called EfficientNets. The models achieve better accuracy and efficiency than previous ConvNets (AlexNet [Krizhevsky et al., 2012b], GoogLeNet [Szegedy et al., 2015], SENet [Hu et al., 2020]).

Another approach is to reduce spatial redundancy in the feature maps. Chen et al. [2019] introduce Octave Convolutions (OctConv) to address this redundancy type. The feature maps are factorised into into two groups at different spatial frequencies and processed with different convolutions at their corresponding frequency, one octave apart. As the resolution for low frequency maps can be reduced, this saves both storage and computation. Similarly to EfficientNets, OctConv improves not only the efficiency, but also the accuracy of the models. The low resolution processing path in the OctConv layer increases the size of the receptive field in the original input space, which is a plausible explanation of the improved performance for image classification.

5.3 Multi-scale octave convolutions

In this chapter, we extend the octave convolution concept to *multi-scale octave convolutional layers*, which include lower resolution feature maps with a higher compression rate (reduction by more than one octave), and the use of more than two feature map tensor groups in order to learn representations at multiple scales[2]. We strive to use multi-scale convolutional representations to learn more robust speech representations.

The motivation for using OctConv for acoustic modelling stems from the fact that some of the feature maps in the CNN model may need to represent information which varies at a lower rate, such as the characteristics of the speaker or background noise, compared to the information necessary for phonetic discrimination. Spatial average pooling in a low resolution group of feature maps can be interpreted as a form of low-pass filtering, providing smoothed representations of the observed data, potentially leading to improved performance. The motivation for using more than two resolution groups stems from the properties of the speech signal – it is rich in spectral content, therefore, learning representations of finer resolution granularity might be beneficial.

Octave convolutions were introduced in the computer vision field to reduce the spatial redundancy of the feature maps by decomposing the output of a convolutional layer into feature maps at two different spatial resolutions[3], one octave apart. This approach improved the efficiency as well as the accuracy of the CNN models. The accuracy gain was attributed to the enlargement of the receptive field in the original input space. We argue that octave convolutions likewise could improve the robustness of learned representations due to the use of average pooling in the lower resolution group, acting as a low-pass filter. We test this hypothesis by evaluating on two noisy speech corpora – Aurora-4 and AMI. We extend the octave convolution concept to multiple resolution groups and multiple octaves.

An octave convolutional layer factorises the output feature maps of a convolutional layer into two groups. The resolution of the low-frequency feature maps is reduced by an octave – height and width dimensions are divided by 2. In this work, we explore spatial reduction by up to 3 octaves – dividing by 2^t,

[2]Note however that the time window for the input features is of a fixed length. Multi-scale learning is realised by down- and up-sampling hidden representations.

[3]The terms *resolution*, *frequency* and *scale* are used inter-changeably in this chapter.

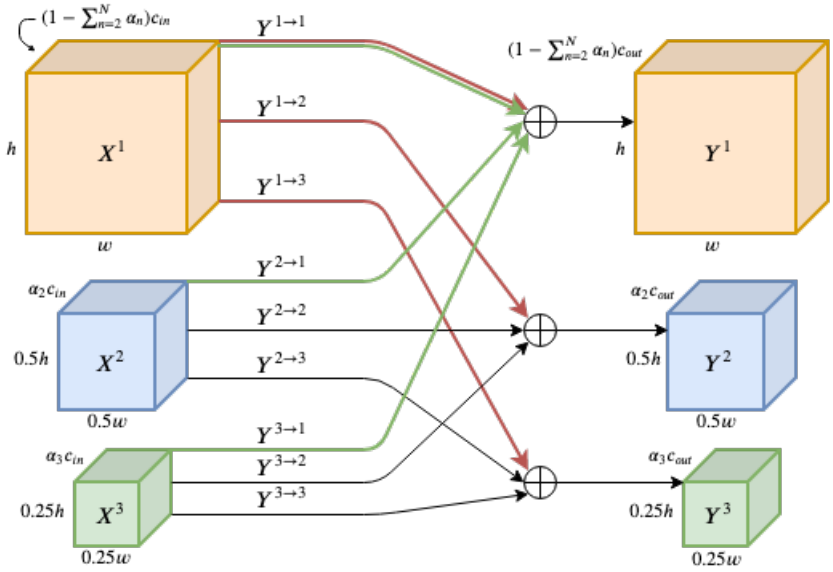


Figure 5.1: Multi-octave convolution scheme for 3 resolution groups. Red and green arrows show the connections for the initial and final MultiOctConv layers, respectively. N corresponds to the total number of groups in the MultiOctConv layer ($N = 3$ in this example). α_n is a fraction of channels corresponding to group n. h and w are spatial dimensions.

where $t = 1, 2, 3$ – and for up to 4 groups. We refer to such a layer as a multi-octave convolutional (MultiOctConv) layer, and an example with three groups and reductions of one and two octaves is depicted in Figure 5.1.

In a vanilla CNN the convolutions have the same spatial resolution throughout the network. The value of the hidden feature at i-th and j-th position in a feature map y can be defined as:

$$y_{i,j} = \psi(\sum_{k=0}^{m-1}\sum_{\ell=0}^{m-1} w_{k,\ell} x_{i+k,j+\ell} + b) \tag{5.1}$$

where ψ is the activation function, $w_{k,\ell}$ are the elements of a weight matrix $\boldsymbol{w} \in \mathbb{R}^{m\times m}$ shared across a feature map, b is the shared bias, and $x_{i+k,j+\ell}$ is the input at $i+k, j+\ell$, with k, ℓ being the indexes of the $m \times m$ receptive field. The parameters in a vanilla convolutional layer are therefore shared across spatial dimensions of each feature map, but not across channels – there is a unique kernel for each input channel.

An octave convolutional (OctConv) layer is divided into high- and low-frequency feature maps and a multi-octave convolutional (MultiOctConv) layer

has feature maps reduced by multiple octaves. Let the input feature tensor be $X \in \mathbb{R}^{c_{in} \times h \times w}$, where c_{in} denotes the number of input channels and h and w correspond to the spatial dimensions[4]. In a MultiOctConv layer working at 3 resolutions, X is factorised along the channel dimension into $X = \{X^1, X^2, X^3\}$. The first group tensor, X^1, is a representation at the same spatial scale as X. The spatial dimensions of the second and third group tensors, X^2 and X^3, are reduced by one and two octaves respectively. The feature maps are not evenly distributed across resolution groups – the number of parameters for each resolution group is controlled by a hyper-parameter $\alpha_n \in [0, 1]$, where $\sum_{n=1}^{N} \alpha_n = 1$ for N resolution groups in the MultiOctConv layer. It corresponds to the fraction of the channels in a vanilla Conv layer. For simplicity, we use the same α_n for input and output representations within the same scale group. The dimensions of the input tensors X^1, X^2 and X^3 are described in Figure 5.1.

Similarly, the output tensors are also factorised into $Y = \{Y^1, Y^2, Y^3\}$. Their dimensions are analogous to the dimensions of the input tensors and are described in Figure 5.1. To compute Y^1, Y^2 and Y^3 we operate directly on the factorised input tensors X^1, X^2 and X^3. The information update is implemented as a sum of feature maps from different resolution groups. To be able to sum those representations for a desired output scale, the spatial dimensions of the input tensors must be the same. For this reason, two operations are employed: spatial average pooling for down-sampling ($\downarrow_p$) and bi-linear interpolation for up-sampling ($\uparrow_u$). u is the up-sampling factor, and p is the pooling factor.

The value of the hidden feature at i-th and j-th position in each feature map y in the corresponding resolution group is defined as:

$$\begin{aligned} y^1_{i,j} =& y^{1\to1}_{i,j} + y^{2\to1}_{i,j} + y^{3\to1}_{i,j} \\ =& \psi(\sum_{k=0}^{m-1}\sum_{\ell=0}^{m-1} w^{1\to1}_{k,\ell} x^1_{i+k,j+\ell} + b) \\ &+ \psi((\sum_{k=0}^{m-1}\sum_{\ell=0}^{m-1} w^{2\to1}_{k,\ell} x^2_{i+k,j+\ell} + b)_{\uparrow_2}) \\ &+ \psi((\sum_{k=0}^{m-1}\sum_{\ell=0}^{m-1} w^{3\to1}_{k,\ell} x^3_{i+k,j+\ell} + b)_{\uparrow_4}) \end{aligned} \tag{5.2}$$

[4]For acoustic modelling, the spatial pattern of the input feature tensor to the first OctConv layer corresponds to a time/frequency space. For further layers, the spatial pattern is over abstract representations, thus we refer to it as spatial modelling (however it might have a different interpretation).

$$
\begin{aligned}
y^2_{i,j} =& y^{1\to 2}_{i,j} + y^{2\to 2}_{i,j} + y^{3\to 2}_{i,j} \\
=& \psi(\sum_{k=0}^{m-1}\sum_{\ell=0}^{m-1} w^{1\to 2}_{k,\ell} x^1_{\downarrow_2, i+k, j+\ell} + b) \\
&+ \psi(\sum_{k=0}^{m-1}\sum_{\ell=0}^{m-1} w^{2\to 2}_{k,\ell} x^2_{i+k, j+\ell} + b) \\
&+ \psi((\sum_{k=0}^{m-1}\sum_{\ell=0}^{m-1} w^{3\to 2}_{k,\ell} x^3_{i+k, j+\ell} + b)_{\uparrow_2})
\end{aligned}
\tag{5.3}
$$

$$
\begin{aligned}
y^3_{i,j} =& y^{1\to 3}_{i,j} + y^{2\to 3}_{i,j} + y^{3\to 3}_{i,j} \\
=& \psi(\sum_{k=0}^{m-1}\sum_{\ell=0}^{m-1} w^{1\to 3}_{k,\ell} x^1_{\downarrow_4, i+k, j+\ell} + b) \\
&+ \psi(\sum_{k=0}^{m-1}\sum_{\ell=0}^{m-1} w^{2\to 3}_{k,\ell} x^2_{\downarrow_2, i+k, j+\ell} + b) \\
&+ \psi(\sum_{k=0}^{m-1}\sum_{\ell=0}^{m-1} w^{3\to 3}_{k,\ell} x^3_{i+k, j+\ell} + b)
\end{aligned}
\tag{5.4}
$$

In the above equations the index of the down or up arrow corresponds to the scaling factor. Down-sampling is performed *before* the convolution, and up-sampling *after* the convolution. The order of all operations for down-sampling is *pooling* $\rightarrow$ *conv2D* $\rightarrow$ *act. func.* $\rightarrow$ *batch norm.*, whereas for up-sampling it is *conv2D* $\rightarrow$ *interpolation* $\rightarrow$ *act. func.* $\rightarrow$ *batch norm.*. By using this order, the convolutions are performed more efficiently.

We will explain up-sampling and down-sampling for a scaling factor of 2. For up-sampling, we use bi-linear interpolation instead of nearest neighbour interpolation [Chen et al., 2019] or zero-order interpolation [Allebach, 2005]. For nearest neighbour interpolation, the value in the up-sampled feature map $x_{\uparrow_u}(.)$, given a value in the original feature map $x(.)$, is defined as

$$
x_{\uparrow_u}(i,j) = x(\lfloor i/u \rceil, \lfloor j/u \rceil) \tag{5.5}
$$

where $\lfloor . \rceil$ denotes rounding to the nearest integer, u is a scaling factor and i and j are the coordinates of points. For $u = 2$ this operation is equivalent to the replication of the nearest value of the feature map $x(.)$ in both dimensions. It is therefore computationally efficient but yields "sharp" representations[5].

[5] A "sharp" representation is a representation which was not smoothed.

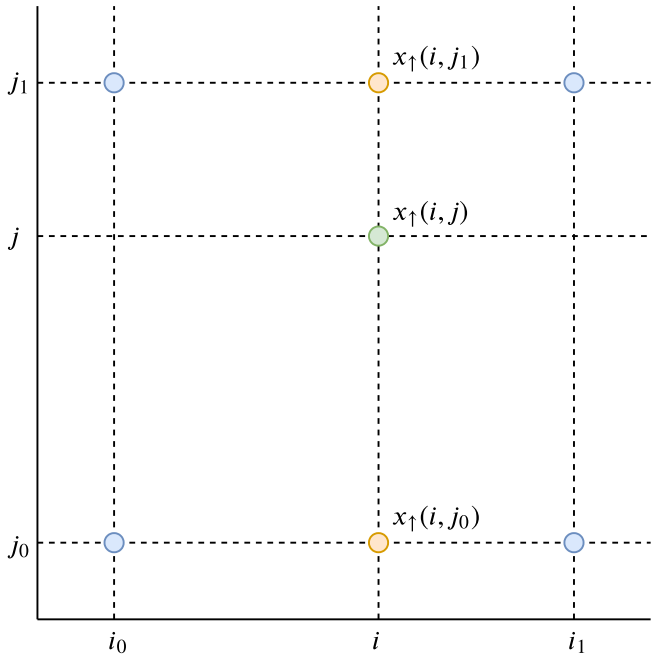


Figure 5.2: Geometrical illustration of bi-linear interpolation. Blue points are the values in the original feature map. Green point is in the up-sampled feature map, at which we interpolate.

To consider the values of four neighbouring points ($x(i_0, j_0)$, $x(i_0, j_1)$, $x(i_1, j_0)$, $x(i_1, j_1)$), and therefore obtain a smoother result, one can linearly interpolate between adjacent samples in two dimensions with bi-linear interpolation [Bovik, 2009]. See Figure 5.2 for the geometrical illustration. First, linear interpolation in i direction yields

$$x_{\uparrow_2}(i, j_0) \approx \frac{i_1 - i}{i_1 - i_0} x(i_0, j_0) + \frac{i - i_0}{i_1 - i_0} x(i_1, j_0), \tag{5.6}$$

$$x_{\uparrow_2}(i, j_1) \approx \frac{i_1 - i}{i_1 - i_0} x(i_0, j_1) + \frac{i - i_0}{i_1 - i_0} x(i_1, j_1). \tag{5.7}$$

Then, linear interpolation in j direction gives the final estimate (green point):

$$x_{\uparrow_2}(i, j) \approx \frac{j_1 - j}{j_1 - j_0} x_{\uparrow_2}(i, j_0) + \frac{j - j_0}{j_1 - j_0} x_{\uparrow_2}(i, j_1). \tag{5.8}$$

A value of a new point is a distance-weighted average. For the neighbouring points at equal distance from the computed value, the new value is simply an average of four neighbouring points.

The down-sampling is realised as 2D average pooling. For a pooling factor $p = 2$, the shift of 0.5 results in non-integer indexes of a down-sampled feature map:

$$x_{\downarrow_2}(i, j) = x(2i + 0.5, 2j + 0.5) \tag{5.9}$$

The values at non-integer locations are simply the averages of the two closest values at integer locations. Therefore, average pooling is a smooth operation, similarly to bi-linear interpolation. An analogous "sharp" operation to nearest neighbour interpolation would be max pooling.

In our implementation, spatial average pooling is simply an average pooling layer $\texttt{pool}(X, p)$ and bi-linear interpolation is up-sampling $\texttt{upsample}(X, u)$, where p is the kernel size and stride for the 2D pooling layer and u is the up-sampling factor. The output MultiOctConv representations are therefore computed as

$$\begin{aligned}
Y_1 &= f(X^1; W^{1\to1}) + \texttt{upsample}(f(X^2; W^{2\to1}), 2) + \texttt{upsample}(f(X^3; W^{3\to1}), 4) \\
Y_2 &= f(X^2; W^{2\to2}) + \texttt{upsample}(f(X^3; W^{3\to2}), 2) + f(\texttt{pool}(X^1, 2); W^{1\to2}) \\
Y_3 &= f(X^3; W^{3\to3}) + f(\texttt{pool}(X^1, 4); W^{1\to3}) + f(\texttt{pool}(X^2, 2); W^{2\to3})
\end{aligned}$$

where $f(.)$ is the convolution function and $W^{n_{in}\to n_{out}} \in \mathbb{R}^{c_{in}\times k\times k\times c_{out}}$ is the convolution filter for a $k \times k$ kernel. We call the information update "intra-frequency" when the number of groups $n_{in} = n_{out}$, and "inter-frequency" when $n_{in} \neq n_{out}$. Note that the convolution $f(.)$ operates on the tensors compressed with average pooling and on the tensors before up-sampling, making the design more efficient. The number of parameters in the MultiOctConv layer is the same as in a vanilla convolutional layer.

The theoretical computational cost and memory footprint gains for Multi-OctCNNs are estimated in Appendix A. This insight into the efficiency is an indication of the model's speed *before* training, which is of great practical value. The computational cost (i.e. the number of operations) is lower for the OctCNN and MultiOctCNN models than for vanilla CNNs. The number of operations depends on the number of groups in the factorised layer, the spatial compression rate, and number of channels for each group. The most computationally efficient model in this chapter uses 54% of the computations of a vanilla CNN with the

same number of parameters. The training speed also depends on the memory footprint, which is reduced with multi-scale models. The lowest memory footprint model in this chapter make up 77% of memory of a corresponding vanilla CNN.

5.4 Experiments

We evaluate our models on multi-condition Aurora-4 and AMI datasets. For AMI experiments, we train our models using the MDM data and evaluate the models for all 3 types of recordings to analyse the effect of mismatched training/testing conditions. We use 40-dimension Mel-scaled Filterbank (FBANK) features with {-5,..,5} context for splicing, resulting in a 40×11 input feature map. Our baseline CNN model [Rownicka et al., 2017] consists of 15 convolutional and one fully-connected layer (Table 5.1). We use 3×3 kernels throughout the network. We start with 64 output channels in the first layer and double them after 3 and 9 layers. We use batch normalization in every convolutional layer, and ReLU afterwards (unless a reverse order is noted). The initial learning rate is 0.001. We use early stopping for training. We use Tensorflow 1.4.0 framework with CUDA 8.0 computing platform and cuDNN 6.0 library version for all experiments in this chapter.

We present our results in terms of accuracy on Aurora-4 and AMI, as well as in terms of the computational and memory cost. Computational cost is calculated as the number of multiply-accumulate operations (MACCs) performed for a single input feature map, and the memory cost is calculated as the memory needed for the output representations. The cost reduction when using octave convolutions stems from reduced dimensions c_{in}, c_{out}, h, and w compared to a vanilla convolutional layer.

5.4.1 CNN vs. OctCNN vs. MultiOctCNN

To compare a vanilla Conv layer with factorised OctConv and MultiOctConv layers, we use the Aurora-4 and AMI datasets. We use Aurora-4 to verify if our methods are effective for noisy speech recognition and in a mismatched microphone scenario. For the experiments with the AMI dataset, we use the MDM training set and evaluate on IHM, SDM and MDM, to assess if the proposed

layer (L)	c_{in}	$h_{in} \times w_{in}$	c_{out}	$h_{out} \times w_{out}$
Conv1	1	40x11	64	40x11
Conv2	64	40x11	64	40x11
Conv3	64	40x11	64	20x11
Conv4	64	20x11	128	20x11
Conv5	128	20x11	128	20x11
Conv6	128	20x11	128	10x11
Conv7	128	10x11	128	10x11
Conv8	128	10x11	128	10x11
Conv9	128	10x11	128	5x6
Conv10	128	5x6	256	5x6
Conv11	256	5x6	256	5x6
Conv12	256	5x6	256	3x3
Conv13	256	3x3	256	3x3
Conv14	256	3x3	256	3x3
Conv15	256	3x3	256	2x2
FC	256	2x2	3422	1x1

Table 5.1: Baseline CNN network structure. c_{in}, c_{out} are the number of input and output channels for Conv layers, respectively (c_{out} for a fully-connected (FC) layer is the number of units). h_{in}, h_{out}, w_{in}, w_{out} are the input and output dimensions (height and width) of the feature maps.

approaches can learn more robustly from reverberant data and generalise well to mismatched acoustic conditions.

The main results for Aurora-4 are presented in Table 5.2. We replace vanilla convolutional layers of our baseline model (CNN) with OctConv and MultiOctConv layers. We first evaluate which layers can be replaced and find that all but the first layer, operating directly on the input representation, should be replaced for the best performance. This approach (L2-L15) is also the least costly. Reducing the ratio of low-resolution representations to 0.125 improves the WER for the mismatched microphone scenario C, but not for all test conditions. Applying batch normalization after ReLU is beneficial for test set C and D. For OctCNN models, the WER for test set D dropped by $\sim 0.4\%$ with a compression by one octave, and by another $\sim 0.4\%$ with a reversed batch normalization and ReLU order.

The main differences between the MultiOctCNN models can be observed for test set D. The models with the lowest WERs are the ones with a spatial reduction of 2 or 3 octaves, and with 2 or 3 groups. This indicates that multi-scale octave

Model	OctConv	α (low $\rightarrow$ high)	2^1	2^2	2^3	#MACCs (M)	Memory (MB)	WER				
								A	B	C	D	Avg.
CNN	-	[0, 1]	-	-	-	174.7	0.798	2.19	4.68	4.22	14.53	8.69
OctCNN	L1-L3	[0.2, 0.8]	✓	-	-	167.6	0.753	2.19	4.74	4.32	14.83	8.85
OctCNN	L1-L15	[0.2, 0.8]	✓	-	-	126.9	0.679	2.22	4.61	4.30	14.40	8.61
OctCNN	L2-L15	[0.2, 0.8]	✓	-	-	126.2	0.695	2.02	4.65	4.35	14.16	8.52
OctCNN †	L2-L15	[0.2, 0.8]	✓	-	-	126.2	0.695	2.22	4.82	4.22	13.72	8.41
OctCNN	L2-L15	[0.125, 0.875]	✓	-	-	143.1	0.734	2.11	4.56	4.07	14.55	8.63
MultiOctCNN	L2-L15	[0.1, 0.1, 0.8]	✓	✓	-	120.6	0.683	**1.98**	4.51	4.11	14.00	8.37
MultiOctCNN	L2-L15	[0.1, 0.1, 0.8]	✓	-	✓	119.5	0.680	2.02	4.59	**3.92**	13.82	**8.31**
MultiOctCNN †	L2-L15	[0.1, 0.1, 0.8]	✓	-	✓	119.5	0.680	2.28	4.81	4.04	13.76	8.41
MultiOctCNN	L2-L15	[0.1, 0.1, 0.1, 0.7]	✓	✓	✓	**94.3**	**0.616**	2.30	4.88	4.18	14.06	8.58
MultiOctCNN	L2-L15	[0.2, 0.8]	-	✓	-	115.7	0.670	2.15	4.77	4.07	13.77	8.39
MultiOctCNN	L2-L15	[0.125, 0.875]	-	✓	-	136.3	0.718	2.09	4.56	4.22	14.32	8.54
MultiOctCNN	L2-L15	[0.2, 0.8]	-	-	✓	113.5	0.665	2.09	4.54	3.94	14.03	8.39
MultiOctCNN	L2-L15	[0.125, 0.875]	-	-	✓	134.9	0.715	2.02	**4.50**	4.17	13.87	8.32
MultiOctCNN †	L2-L15	[0.125, 0.875]	-	-	✓	134.9	0.715	2.32	4.73	4.24	**13.57**	**8.31**

Table 5.2: WERs [%] for Aurora-4 test sets A, B, C and D for octave and multi-octave CNNs. "OctConv" column indicates where a Conv layer was replaced with an OctConv or MultiOctConv. 2^1, 2^2 and 2^3 correspond to the factor of spatial dimension reduction. Models with batch normalization after ReLU are denoted by †.

Model	OctConv	α (low $\rightarrow$ high)	2^1	2^2	2^3	#MACCs (M)	Memory (MB)	WER					
								IHM		SDM		MDM	
								dev	eval	dev	eval	dev	eval
CNN	-	[0, 1]	-	-	-	175.2	0.800	33.4	38.3	49.1	54.0	43.9	48.0
OctCNN	L1-L3	[0.2, 0.8]	✓	-	-	168.2	0.755	33.0	38.1	49.0	54.1	43.8	47.9
OctCNN	L2-L15	[0.2, 0.8]	✓	-	-	126.7	0.698	33.0	37.7	48.9	54.0	43.7	47.7
OctCNN	L1-L15	[0.2, 0.8]	✓	-	-	127.5	0.681	**32.2**	**37.2**	48.3	53.5	43.1	47.3
OctCNN	L1-L15	[0.125, 0.875]	✓	-	-	144.1	0.726	32.5	37.4	**48.2**	**53.3**	**42.9**	**47.2**
OctCNN†	L1-L15	[0.125, 0.875]	✓	-	-	144.1	0.726	33.2	38.3	48.8	54.3	43.7	48.0
MultiOctCNN	L1-L15	[0.1, 0.1, 0.8]	✓	✓	-	121.6	0.666	32.8	38.1	48.9	53.9	43.7	47.9
MultiOctCNN	L1-L15	[0.1, 0.1, 0.8]	✓	-	✓	120.4	0.663	33.3	38.5	49.2	54.5	44.1	48.4
MultiOctCNN	L1-L15	[0.1, 0.1, 0.1, 0.7]	✓	✓	✓	**95.2**	**0.588**	33.7	38.7	49.5	54.6	44.1	48.4
MultiOctCNN	L1-L15	[0.125, 0.875]	-	✓	-	136.9	0.707	33.6	38.6	49.7	54.6	44.3	48.4
MultiOctCNN	L1-L15	[0.125, 0.875]	-	-	✓	135.4	0.703	32.9	38.1	49.1	54.3	43.8	48.0

Table 5.3: WERs [%] for models trained on AMI MDM and evaluated on IHM, SDM and MDM conditions. "OctConv" column indicates where a Conv layer was replaced with an OctConv or MultiOctConv. 2^1, 2^2 and 2^3 correspond to the factor of spatial dimension reduction. Models with batch normalization after ReLU are denoted by †.

convolutions seem to be an effective, as well as an efficient, design for processing speech with background noise and channel mismatch. For MultiOctCNNs, batch normalization after ReLU also gives a performance boost for test set D, with a drop to 13.57%.

The main results for AMI are presented in Table 5.3. In contrast to the Aurora-4 findings, better performance was achieved with an all OctCNN model (L1-L15). This is an interesting finding, and we believe that the multi-scale processing of the input feature space is beneficial for AMI MDM because of the reverberation in the data. The reverberant input time×freq representation can be viewed as a spatially redundant one, therefore the OctConv layer applied to the input representation is effective. The only MultiOctConv model superior to the baseline CNN is the one with 3 groups with a spatial reduction by 1 and 2 octaves. This result indicates that the spatial redundancy for this architecture for AMI MDM is not degrading the performance. However, in terms of the computational cost, we can reduce the #MACCs by a factor of 1.8 with only a small WER increase for a model with 4 resolution groups. This model also uses 1.4 times less memory for the output representations.

5.4.2 Layer-dependent granularity

The α hyper-parameter controls the ratio of multi-scale feature maps in the model, enabling both learning of fine-grained representations preserving the details necessary for phonetic discrimination, as well as smoothed representations improving the robustness of the model. Setting α layer-by-layer to enable the fractions of channels at different resolutions to vary according to the depth of the model would enable controling the granularity of the representations learned at every layer. We evaluate how does the gradual change of α for low resolution representations influences the WERs in our best MultiOctCNN model (last row in Table 5.2) and we present the results for Aurora-4 in Table 5.4.

By altering the ratio of low and high resolution channels independently for the convolutional layers, we were able to decrease the WER to 8.23%, with 2.32%, 4.81%, 4.18%, 13.31% for test sets A, B, C, and D, respectively. The result for test set D indicates, that this approach is effective especially for learning noise robust speech representations. The optimal ratio of the channels was 0.5 in the initial layers (i.e. the same number of channels was used for smoothed representations,

$\alpha_{low}^{(2-3)}$	$\rightarrow \alpha_{low}^{(4-6)}$	$\rightarrow \alpha_{low}^{(7-9)}$	$\rightarrow \alpha_{low}^{(10-12)}$	$\rightarrow \alpha_{low}^{(13-15)}$	WER
0.125	$\rightarrow$ 0.125	$\rightarrow$ 0.125	$\rightarrow$ 0.125	$\rightarrow$ 0.125	8.31
0.9	$\rightarrow$ 0.7	$\rightarrow$ 0.5	$\rightarrow$ 0.3	$\rightarrow$ 0.1	9.67
0.7	$\rightarrow$ 0.55	$\rightarrow$ 0.4	$\rightarrow$ 0.25	$\rightarrow$ 0.1	8.76
0.5	$\rightarrow$ 0.4	$\rightarrow$ 0.3	$\rightarrow$ 0.2	$\rightarrow$ 0.1	**8.23**

Table 5.4: Results for Aurora-4 for α_{low} (fraction for the low resolution group) changing gradually across the layers.

as well as for the detailed ones), dropping gradually to 0.1 in the final layers (i.e. only 10% of the final representations are smoothed). It is also interesting to note that by using a higher ratio for low resolution representations is further reducing the computational and memory cost compared to a previously proposed MultiOctConv layer with constant α across the model. Starting with a higher ratio for smoothed representations can be helpful not only for efficiency but also for improved robustness and accuracy; it imposes a constraint to learn more general representations at the lower layers of a model. However, starting with a too high ratio can cause the representations to loose important information, which can not be recovered by the higher layers. We believe this is the reason for the degradation in performance for the models with 0.9 and 0.7 as the initial α value.

We have also tested this approach for distant speech recognition task (AMI MDM), however, we did not observe the gains in performance. Further layer-by-layer tuning of the α hyper-parameter in the MultiOctConv design is needed to find the optimal setting for this task.

5.4.3 Inter-frequency exchange paths

Chen et al. [2019] show that inter-frequency connectivity brings accuracy gains in an OctCNN model for computer vision. We also evaluate if those connections are crucial for the MultiOctCNN design. We disable the information exchange between different resolution groups, such that there is only one path left for each resolution group in a MultiOctConv layer. For a layer with three resolution groups, the representations are therefore simply computed as

$$Y_1 = f(X^1; W^{1\to1})$$
$$Y_2 = f(X^2; W^{2\to2})$$
$$Y_3 = f(X^3; W^{3\to3})$$

where X^2 is a representation one octave apart from X^1, and X^3 is two octaves apart from X^1. The results for Aurora-4 are presented in Table 5.5. These preliminary results are inconclusive – we have observed an improvement for one out of three models when the inter-frequency paths were disabled.

Model	A	B	C	D	Avg.
OctCNN (2 groups, $\alpha_{low} = 0.125$)	1.91	4.68	4.18	14.01	8.45
OctCNN (2 groups, $\alpha_{low} = 0.125$) †	2.15	4.75	4.05	13.69	8.35
MultiOctCNN (2 groups, $\alpha_{low} : 0.5 \to 0.1$)	2.32	4.81	4.18	13.31	8.23
MultiOctCNN (2 groups, $\alpha_{low} : 0.5 \to 0.1$) †	2.24	4.98	4.07	13.85	8.52
MultiOctCNN (4 groups)	2.30	4.88	4.18	14.06	8.58
MultiOctCNN (4 groups) †	2.24	4.89	4.07	14.22	8.64

Table 5.5: Comparison of the Aurora-4 models with disabled inter-frequency connections in a MultiOctConv layer, denoted by †.

For AMI, we also did not observe a big difference in WERs (Table 5.6). Although further experiments are needed to confirm that disabling the inter-frequency exchange paths is not harmful for different models, the preliminary results indicate that a similar performance can be achieved with only one connection per resolution group. By disabling the inter-frequency connections, the input tensor will only be read once per group, reducing the total number of memory accesses. The number of computations and the number of parameters will also be reduced. With this approach, the representations are still learned in a multi-scale fashion and combining the representations at different resolutions in a final MultiOctConv layer (green arrows in Figure 5.1) seems to be sufficient for learning robust speech representations, as well as being more computationally and memory efficient.

	IHM		SDM		MDM	
Model	dev	eval	dev	eval	dev	eval
MultiOctCNN (4 groups)	33.7	38.7	49.5	54.6	44.1	48.4
MultiOctCNN (4 groups) †	33.2	38.3	49.3	54.5	44.0	48.5

Table 5.6: Comparison of the models trained on AMI MDM training set with disabled inter-frequency connections in a MultiOctConv layer, denoted by †.

5.4.4 Interpolation type

Next, we evaluate the influence of the type of interpolation on the accuracy. In our experiments, we used the bi-linear interpolation algorithm to up-sample the input representations. We compare it with nearest neighbour and bi-cubic interpolation algorithms.

Interpolation type	A	B	C	D	Avg.
Nearest neighbour	2.20	4.80	4.32	13.63	8.36
Bi-linear	2.32	4.73	4.24	13.57	8.31
Bi-cubic	2.24	4.85	4.26	13.34	8.26

Table 5.7: Comparison of the interpolation type effect for Aurora-4. The models in the table are MultiOctCNNs with 2 groups at $[0.125, 0.875]$ ratio and 2^3 spatial reduction.

All of the approaches give similar performance, with a slight advantage of the bi-cubic interpolation algorithm, especially for test set D. It gives the smoothest representations, potentially contributing to the best performance, however it is significantly slower – it considers 16 locations (4×4) instead of 4 (2×2) as in the bi-linear interpolation. At training, the bi-cubic interpolation (implemented with Tensorflow's `tf.image.resize_images` method) took around 15.9× more time than bi-linear interpolation. At inference (measured as the time needed to write out the log-likelihoods) the bi-cubic interpolation was around 2.8× slower. For this reason, we only evaluate the influence of the interpolation type for a much smaller Aurora-4 corpus. The speed for bi-linear and nearest neighbour algorithms is comparable – the bi-linear interpolation took around 1.1× more time, both at training and at inference.

5.4.5 Alternative input representation

In the experiments so far, we have used 40-dimensional mel-scaled filterbank (FBANK) features. We hypothesise that our multi-scale design could take the advantage of an alternative, more detailed input representation. We first experiment with wider time context, then with a higher frequency resolution for FBANKs extraction, and finally with spectrogram features as the input representation for our multi-scale CNN models.

For AMI, we use IHM training set, not MDM as in previous experiments in this chapter. By training on IHM data we aim to disentangle the evaluation of learning robustly from reverberated data and the evaluation of generalising well to mismatched acoustic conditions. In the experiments so far, multi-scale convolutional representation learning was not as effective in the AMI task as it was for Aurora-4. By training on the better quality recordings (IHM), we aim to leave out the effect of learning from very challenging distant microphone recordings, but we still evaluate on all three conditions (IHM, SDM, MDM), to assess the ability to generalise to more difficult acoustic scenes.

5.4.5.1 Wider time context

Instead of five frames of context for each time step, we evaluated the models with seven and nine frames at both sides (with the same resolution in frequency)[6]. The results are presented in Table 5.8.

Interestingly, a wider time context in general was not advantageous. For most of the models, it was harmful when evaluated on SDM and MDM conditions. A slight accuracy improvement was observed for wider context in Oct and MultiOctCNNs in matched testing conditions (IHM), showing the capability of the factorised design to take the advantage of the increased receptive field. However, the wider time-span models do not generalise very well to new SDM and MDM conditions. Models with five frames of context were optimal in our experiments.

The OctCNN and MultiOctCNN models trained on AMI IHM training set were in general better when the factorised Conv layer was not applied to the input representation (when tested on matched IHM condition). The same effect was observed for the Aurora-4 dataset. It can be explained by the fact that those two training datasets are not corrupted by reverberation, thus the input

[6]Nine was maximum without exceeding the memory and without altering the architecture.

Model	Context	IHM dev	IHM eval	SDM dev	SDM eval	MDM dev	MDM eval
CNN	5	25.4	27.7	59.6	68.4	50.1	57.1
CNN	7	25.4	27.4	62.0	71.5	51.8	58.9
CNN	9	25.6	27.7	61.2	70.2	50.9	57.9
OctCNN $^{L1-L15,[0.2,0.8]}$	5	25.9	28.4	59.6	68.7	50.3	57.4
OctCNN $^{L1-L15,[0.2,0.8]}$	9	25.9	27.8	61.1	70.4	51.0	57.9
OctCNN $^{L2-L15,[0.2,0.8]}$	5	25.8	28.2	59.8	69.1	50.4	57.5
OctCNN $^{L2-L15,[0.2,0.8]}$	7	25.9	27.8	62.1	71.4	51.5	58.6
OctCNN $^{L2-L15,[0.2,0.8]}$	9	25.5	27.6	61.7	70.9	51.1	58.1
MultiOctCNN $^{L2-L15,[0.2,0.8],2^3}$	7	26.0	28.0	62.6	72.0	51.9	59.1
MultiOctCNN $^{L2-L15,[0.2,0.8],2^3}$	9	25.6	27.5	60.4	69.4	50.4	57.1

Table 5.8: Comparison of the AMI models with wider time context. *Context* column corresponds to the number of neighbouring time frames. The models were trained on AMI IHM training set.

representation (40×11 FBANK feature map) is not redundant along the time axis. Therefore, learning from all input features, without smoothing in the first layer, is optimal for AMI IHM and for Aurora-4 datasets.

5.4.5.2 Higher frequency resolution

High resolution MFCC features with 80 bins and cepstra (no dimensionality reduction) were previously explored in an AMI TDNN Kaldi recipe[7] with the improvements in accuracy. We evaluate if the factorised convolutional models can also leverage higher frequency resolution of the input representation. Following the optimal number of the triangular mel-frequency filters applied to the power spectrum in high resolution MFCC extraction, we extract 80-dimensional FBANK features. We use five time frames of left and right context. The results for Aurora-4 are presented in Table 5.9. Due to the out of memory condition when using higher frequency resolution, we used a reduced number of channels (64 for Conv1-6, 128 for Conv7-12, 256 for Conv13-15) for the models in the table above, compared to all previous models presented in this chapter. We believe this is the reason for degraded performance in general. The other configurations for

[7] egs/ami/s5b/local/chain/tuning/run_tdnn_1j.sh

Model	A	B	C	D	Avg.
CNN	2.69	5.13	7.08	15.48	9.53
OctCNN$^{L2-L15,[0.125,0.875],2^3}$	2.91	5.60	6.02	15.05	9.49
MultiOctCNN (2 groups, $\alpha_{low} : 0.5 \rightarrow 0.1$)	2.88	5.43	6.59	15.10	9.48

Table 5.9: Comparison of the Aurora-4 models trained with 80D FBANK features.

the OctCNN model match the best model from Table 5.2; for the MultiOctCNN – the best model from Table 5.4.

The average differences in WERs are small, hence the factorised Conv layer did not leverage higher frequency resolution of the input representation very well. However, a small improvement can be observed for test sets C and D. More experimental results would be necessary to make more general conclusions, however, training with the representations of higher dimensionality is costly, and those preliminary results do not foreshadow big performance gains.

5.4.6 Multi-condition training

Another direction worth exploring in the context of MultiOctCNN models is data augmentation, i.e. multi-condition training. The premise of a multi-scale representation is that it can be more robust than a single-scale one. Multi-scale approaches aim to extract complementary information by learning at multiple scales. We hypothesise that training on an augmented training set, e.g. subjected to different noise types or reverberation, could fit in well with multi-scale processing. If the low resolution representation is more homogeneous regardless of the nuisance factor, the model should be able to leverage this smoothed representation to generalise well to unseen conditions. At the same time, the high resolution representation provides a detailed description of senones in multiple acoustic conditions, reducing the mismatch between training and test conditions. So, we investigated if a multi-scale CNN model can leverage its multi-resolution design in multi-condition training. We report the results in Table 5.10. The data augmentation involved:

- 3-way speed perturbation (using factors of 0.9, 1.0 and 1.1),
- volume perturbation (volume factor chosen randomly from a uniform distribution between 0.125 and 2.0),

- reverberation (using simulated RIRs [Ko et al., 2017] from small and medium rooms and point-source noises extracted from the MUSAN corpus [Snyder et al., 2015]).

	IHM		SDM		MDM	
Model	dev	eval	dev	eval	dev	eval
CNN	26.6	29.2	49.4	54.0	44.6	48.8
OctCNN$^{L1-L15,[0.2,0.8]}$	27.2	30.0	50.2	55.1	45.5	49.5
OctCNN$^{L1-L15,[0.125,0.875]}$	26.0	28.5	48.8	53.6	44.2	48.2
OctCNN$^{L1-L15,[0.125,0.875]\dagger}$	26.0	28.4	48.8	53.6	44.0	48.1
MultiOctCNN$^{L1-L15,[0.1,0.1,0.8]}$	26.6	29.3	49.2	54.3	44.6	48.6

Table 5.10: Results for models trained on augmented AMI IHM training set. Models with batch normalization after ReLU are denoted by †.

The training set was augmented 6 times, resulting in approximately 427 hours in total. The configurations for the models match the best models from Table 5.3. To obtain the alignments for the augmented training set, we first extract fMLLR features for speed and volume perturbed data and we use a pre-trained SAT-GMM triphone model to align the data. Next, we reuse the speed and volume perturbed data alignments for the reverberated data. By doing this, we ensure better quality of the alignments for the reverberated part of the training set.

The WERs for multi-scale models in multi-condition training scenario are not considerably lower than the WER for the baseline model. We believe that although the low resolution representation is smoothed and more homogeneous, it may not necessarily need to similarly represent the same sounds under different distortions. Perhaps, not only the detrimental distortions, but also the information necessary to differentiate between senones is smoothed out, making the ASR task harder. This hypothesis would require more investigation to reach firm conclusions.

5.5 Conclusions

In this chapter, we presented multi-scale octave CNN models for robust and efficient speech recognition. We built on Chen et al. [2019], successfully applying the method to speech recognition and extending it to multiple resolution groups

with a spatial reduction of more than one octave. The proposed method enabled to disentangle the representations by learning about the underlying factors of variation changing at different rates. We also explored layer-dependent granularity of representations, disabling inter-frequency exchange paths, influence of the interpolation type, wider time context and higher frequency resolution of the input representations, and multi-condition training.

Our experiments confirmed that multi-scale processing of the hidden representations is not only more computationally and memory efficient, but also improved the recognition. The proposed method reduced the WER by up to 5.3% relative for Aurora-4 and by up to 3.6% for AMI, while improving the computational and memory efficiency of the CNN acoustic models. Those improvements are significant at the level of $p = 0.001$.

The performance improved especially in the additive noise and mismatched microphone scenario (Aurora-4 corpus). We also observed the gain of the octave convolutions for AMI MDM data with significant reverberation, when applied to the input feature space. However, the model performance for AMI MDM was not improved with multi-octave convolutions. It may be the case that the multi-scale design is favourable for noisy speech but not so much for distant speech recognition. The difference between Aurora-4 and AMI datasets is also in their size, which might be another reason for bigger gains for Aurora-4 than for AMI. Perhaps multi-scale representation learning is especially important when the training set is rather small, in which case an alternative, smoothed representation is complementary and helpful. For a bigger AMI corpus, as well as for the augmented AMI corpus, we observed smaller or no WER reductions.

With layer-dependent ratio of low and high resolution representations we reduced the WER for Aurora-4, but not for AMI MDM. The inter-frequency connectivity turned out to be unnecessary. This finding is in contrast to what was shown by Chen et al. [2019] and needs further investigation; skipping inter-frequency connections further speeded up training and inference. As for the interpolation algorithm, the smoothest bi-cubic interpolation was the most accurate but also the slowest. The results have shown that choosing an up-sampling algorithm which gives a smooth representation is preferable in terms of accuracy. Analogously, for down-sampling, smooth average pooling is preferred over max-pooling. Approximation by smoothing, either to down- or up-sample, was able to capture important patterns in the representations while discarding fine-scale artefacts or

noise.

Wider time context of the input representation was only beneficial in matched testing conditions, but the gains were not substantial. Using too wide a context could contribute to overfitting of the models, which can explain worse performance in mismatched conditions. Using a multi-scale CNN did not prevent the model from overfitting for wider time context for AMI IHM training set. An evaluation for Aurora-4 or AMI MDM training data could be performed to verify this result for noisy and distant speech recognition.

We also evaluated the representations with higher frequency resolution and less processed, spectrogram features. Those experiments were not successful – we did not observe an improvement in accuracy for multi-scale processing using spectrogram features as input representations.

Multi-condition training experiments indicated that a gain from multi-scale representation learning can be achieved but is not significant. Those experiments were time consuming due to much bigger training set, therefore more experiments with a randomly selected subset of utterances from the augmented multi-condition training set could be performed in the future, to test whether the multi-scale CNN models can leverage multi-condition training data better than vanilla CNNs.

In this chapter, we also presented theoretical computations for estimating the computational cost as well as the memory footprint of our models. However, a further analysis of the actual compute workload on the GPU could also be performed in the future. In our theoretical estimations, we assumed a complete reuse of the parameters and the activations. In practice, redundant memory accesses can be realised, depending on the implementation. Also, the choice of the framework and the compute kernel has a big impact on the model speed performance. We propose to use a kernel profiler for CUDA, such as NVIDIA Nsight Compute, to investigate the practical performance metrics. Also, arithmetic intensity (measure of FLOPs relative to the amount of memory accesses) could be used as a single theoretical performance approximation [Liu, 2020].

CHAPTER 6

Analysing learned representations

This chapter is based on [Rownicka et al., 2017], [Rownicka et al., 2018], [Rownicka et al., 2019], and [Rownicka et al., 2020]. The analyses in this chapter aim to improve the understanding of representations learned by acoustic models.

We aim to better understand what makes one representation better than another, by exploring the underlying factors of variation of the input speech signal. We are mostly interested in CNN-based acoustic models with two-dimensional filters, which have proved to be effective for modelling speech[1]. We perform the analyses *post-hoc* – given a trained model, we ask *why* it performs well in terms of accuracy. We strive to interpret learned representations to identify new information about our models, alongside their WER performance. We evaluate the quality of the explanations in the context of this goal. Our main research questions are:

- Which components make a 2D CNN model optimal for acoustic modelling?
- Which underlying factors of variation are informative for speech recognition?
- Can we implicitly learn disentangled representations?

[1]Deep CNN representations analysed in this chapter served as a baseline in previous chapters.

It is important to note that we explore the above questions for specific AMs and specific datasets. One of the main shortcomings in interpretable AI for Deep Neural Networks is *the multiplicity of good models* [Breiman et al., 2001], also referred to as *model locality* [Gill, 2018]. It is a consequence of non-convex error surface (without a single global minimum), which implies that multiple accurate models can be learned, with different sets of parameters. Therefore, from the interpretability perspective, the explanations can only be derived for a particular model. Consistent patterns across different models (with different initial conditions, different architectures, or for different datasets) could provide more stable explanations and could potentially enable generalisation beyond a single model. However, in this chapter, we focus on analysing specific models, that were previously found to perform well in terms of accuracy.

We first provide a brief overview of the methods used in the literature to better understand NN predictions and we review the work in this area for ASR (Section 6.1). We then describe our experiments on simplifying deep CNN architectures in Section 6.2. In Section 6.3 we show local and global visualisations of learned representations, and in Section 6.4 we quantitatively evaluate the representations with a verification framework. In Section 6.5 we use a linear transformation loss as a proxy robustness measure to compare learned representations.

6.1 Background

The focus of previous chapters in this thesis was on improving the accuracy of the acoustic models. In this chapter, we attempt to better understand learned representations by analysing and assessing them with more than WERs. In the literature, *explainable* and *interpretable* ML (or AI) refer to techniques for better model understanding. At the time of writing the thesis, it is a very active research direction. In the paper based on a tutorial given at ICASSP 2017, Montavon et al. [2018] provide definitions for interpretation and explanation in the context of DNNs as follows:

Definition 1. "*An interpretation is the mapping of an abstract concept (e.g. a predicted class) into a domain that the human can make sense of.*"

Definition 2. "*An explanation is the collection of features of the interpretable domain, that have contributed for a given example to produce a decision (e.g. classification or regression).*"

Interpretable domains can be images or text, and examples of non-interpretable ones include word embeddings. One example of an explanation is a saliency map [Itti et al., 2009], showing which regions in the input support the classification decision the most. Explanations concern individual predictions, while interpretation aims to provide a global analysis. Another definition by Kim et al. [2016a] states that "a method is interpretable if a user can correctly and efficiently predict the method's results".

A strong interest of ML community in interpretable AI is therefore motivated by a chance to improve [Doshi-Velez and Kim, 2017; Carvalho et al., 2019]:

- scientific understanding – explanations can be converted into knowledge;
- robustness – more robust models (or models more resistant e.g. to adversarial attacks) can be build based on better model understanding;
- reliability – by predicting model's performance, giving more confidence in certain performance for unseen conditions;
- fairness – by identifying bias in the explanations and designing unbiased models;
- privacy – by ensuring that sensitive information is protected;
- trust – by understanding the reasons for the decisions made, users can have more trust in the predictions.

Our work in this chapter is motivated primarily by better scientific understanding of the representations learned by acoustic models.

6.1.1 Interpretability methods

A number of different approaches have been proposed recently, mostly for Computer Vision applications, to better understand NN models. The approaches can broadly be categorised into *global* and *local*. Global interpretability is aimed at discovering general patterns, which is of interest when scientific understanding

or bias detection is the main goal. Local interpretability focuses on example-based explanations, with the goal to provide justification for a specific decision [Doshi-Velez and Kim, 2017]. Another classification proposed by Olah et al. [2018] distinguises *feature visualisation*, *attribution*, and *dimensionality reduction*.

Feature visualisation methods address the question of what the network has learned, and provide the analysis by generating examples [Olah et al., 2017]. Those methods are also referred to as *activation maximisation* [Montavon et al., 2018]. The goal is to optimise the input such that it produces a maximum response for a quantity of interest, e.g. a neuron firing or the output behaviour. The prototype input can be found with an optimisation technique – by gradient ascent in the input space [Erhan et al., 2009]. For improved interpretability and quality of the prototype, the optimisation process can be regularised as proposed e.g. by Yosinski et al. [2015]. Feature visualisation can pertain to visualising individual neurons, the combination of neurons, or the activations – the combination of neurons firing in response to a particular input. Activation atlases have been proposed by Carter et al. [2019] to visualise features (with an iterative optimisation process) of averaged activations. Gradient-based visualisation methods are related to deconvolutional networks proposed by Zeiler and Fergus [2014]. Simonyan et al. [2013] show that gradient-based visualisation can be seen as the generalisation of deconvolutional networks.

Dimensionality reduction can be used to transform high dimensional representations into low dimensional explanation space to visualise the data. A common method used for this purpose is t-SNE, introduced by van der Maaten and Hinton [2008]. The t-SNE algorithm is non-linear, and uses the whole dataset to perform different transformations in different regions. It also has hyper-parameters: perplexity, which controls attention to local and global patterns in data, and other hyper-parameters, related to the optimisation process. Those two aspects of the algorithm can make the visualisations prone to misinterpretations [Wattenberg et al., 2016]. Nevertheless, t-SNE is a very common technique for visualising high-dimensional representations, and with a careful hyper-parameter investigation and analysis of the plots, it is a meaningful dimensionality reduction technique.

Karpathy [2014] uses t-SNE to arrange CNN codes (4096-dimensional activation vectors extracted from the last fully-connected layer of a CNN model) in a two-dimensional space, and maps the CNN codes back to the input images for visualisation. The resulting plot arranges images according to their CNN code

similarity. This analysis helps to infer how the network is responding to the examples in the dataset. In Section 6.3.2, we perform a similar analysis, with the differences in CNN code extraction. We do not map the codes back to the input representation, but we show the labels for different utterance attributes, for better interpretability.

In Sections 6.4 and 6.5, we also analyse the activations[2] – the combination of neurons and the input data – rather than analysing learned filters, to be able to infer higher-level concepts learned by the network for a specific dataset. The limitation of this approach is the dependency on the distribution of the data we sample the activations from, making the explanations less generic (yet more comprehensible).

Attribution is another class of methods used for interpretation, where the explanations are generated for a specific sample (those methods are also referred to as *sensitivity analysis* [Montavon et al., 2018]). Saliency maps, proposed by Itti et al. [2009] to select attended locations in order of decreasing saliency, are nowadays commonly used for explaining predictions in ML. Simonyan et al. [2013] use saliency maps to show spatial support of a particular class in a given image using back-propagation through a trained CNN-based image classifier. They also propose to use saliency maps for weakly supervised object localisation. Alongside saliency maps, other attribution methods, such as Integrated Gradients [Sundararajan et al., 2017], Pattern Attribution [Kindermans et al., 2018], or Deep Taylor Decomposition [Montavon et al., 2017] were also proposed for Computer Vision applications, to determine the attribution of each input dimension. We refer the reader to [Kindermans et al., 2019] for a comparison of those methods (and a discussion on reliability of saliency methods). Other examples of some of recently proposed frameworks to explain local predictions are LIME [Ribeiro et al., 2016] and SHAP [Lundberg and Lee, 2017]. The authors provide extensive online implementations of the methods, making them easily accessible. Another open-source examples of explainability toolkits are IBM's AI Explainability 360 [Arya et al., 2019], Uber's Manifold (a visual debugging tool) [Uber, 2020], or Captum (model interpretability and understanding toolkit for PyTorch) [Kokhlikyan et al., 2019], demonstrating interest in explainable ML not only in academia, but also in industry.

[2]More specifically, we use pre-activations, i.e. before applying ReLU, such that the distribution of the representations is better suited for further processing.

Ablation studies of the architectures can also be seen as an attribution analysis. Zeiler and Fergus [2014] performed an ablation study to discover the performance contribution from different layers. They found that the depth of the network was the most important factor performance-wise, rather than any individual component. They also found, however, that removing the fully-connected layers resulted in only a slight decrease of the accuracy. This result is surprising and we investigate it further in Section 6.2. Simplifying a typical CNN architecture (with pooling and fully-connected layers) to an architecture consisting solely of convolutional layers has been shown to achieve state-of-the-art accuracy on several image recognition tasks – for instance, GoogLeNet [Szegedy et al., 2015], is a convolutional network which uses average pooling instead of fully-connected layers, eliminating the majority of trainable parameters of the network without reducing object recognition accuracy. Similarly, Springenberg et al. [2015] question the necessity of different components in the network, and show that a fully-convolutional model can yield competitive or state-of-the-art performance on several object recognition datasets. Deep residual learning (ResNet) [He et al., 2016], is another example of a successful deep CNN without fully-connected layers at the top of the network. For the ASR task, Palaz et al. [2015a] investigate a CNN-based framework with a simple linear classifier and show superior performance with this architecture.

Knowledge distillation [Hinton et al., 2015], or *teacher-student* training, where a smaller model is trained using the output of a larger model, can be seen as explainability method. This method proved to be effective also for ASR [Lu et al., 2017; Watanabe et al., 2017; Fukuda et al., 2017]. The goal is to explain a complex NN with a simpler model, so in this sense it can be seen as an explainability approach. However, the underlying interpretations of the predictions are not of interest here, unless a more interpretable surrogate (i.e. student) model is used (e.g. a decision tree).

Comparison of the representations is another interpreatbility approach which can improve the understanding of the models. It is a challenging task since the structure of the representations varies within and between models. Comparison of the representations can be used e.g. to probe learning dynamics throughout training or to show where class-specific information in the networks is formed. Raghu et al. [2017] propose Singular Vector Canonical Correlation Analysis (SVCCA) to compare two representations in a way that is invariant to affine transform,

allowing comparison between different layers and networks. Morcos et al. [2018] develop projection weighted Canonical Correlation Analysis (CCA) and show that networks which generalise converge to more similar representations than networks which memorise. Kornblith et al. [2019] propose to use Centered Kernel Alignment (CKA) to identify correspondences between representations in networks trained from different initialisations, while Thompson et al. [2019] propose to use modified RV-coefficient (RV2), which has similar properties to CKA while being less sensitive to dataset size. Our approach in Section 6.5 also attempts the comparison of the representations. We propose a NN-based method to measure the distance between clean and noisy representations, and use it as an indication of the robustness of learned representations (with a positive correlation of similarity between clean and noisy representations and robustness of the acoustic model).

Another group of methods which can be seen as interpretability is what we refer to as *diagnostic verification*. It is related to diagnostic classification, where the representations to be evaluated are used in another classification task. For instance, Alain and Bengio [2017] train linear classifiers on intermediate representations and show prediction errors from every layer in the network. Raj et al. [2019] probe the information encoded in x-vectors and compare it to the information encoded in i-vectors. The next section gives an overview of other interpretability methods for speech recognition.

6.1.2 Interpretability for ASR

An input to a NN-based acoustic model can be in a form of a raw waveform, or acoustic features such as MFCCs or FBANKs. Acoustic features are speech representations which exploit the knowledge about human auditory system – they are compact, and at the same time, they encode the differences between different sounds. Visually, the acoustic features are not very interpretable. Therefore, the input to an acoustic model is harder to interpret than the input to an image classifier, let alone the latent representations.

Nevertheless, better acoustic model understanding has been addressed previously in the literature. Palaz et al. [2015b, 2019] analyses the information learned by CNNs from raw waveform, and show that the first two layers learn phone-specific spectral envelope information. rahman Mohamed et al. [2012] visualise feature vectors learned by Deep Belief Networks (DBNs) and investigate

which aspects of the model are responsible for its good performance. Nagamine et al. [2016] explore the role of non-linear transformations in DNN acoustic models. They look at sigmoid activation function, and the models are trained for a phoneme recognition task. The authors evaluate how the non-linear transformations are applied to the features when learning categorical boundaries for phonemes. Nagamine et al. [2015] and Nagamine and Mesgarani [2017] analyse the activations of a DNN acoustic model and cluster the activations, also according to monophone labels. Krug and Stober [2018] and Krug et al. [2018] perform a similar analysis – clustering class-specific activations – but for fully-convolutional acoustic models. Krug and Stober [2020] propose a method to show characteristic responses of the model to particular groups of inputs, which incorporate the relevance of neurons for the prediction. [Muckenhirn et al., 2019] propose a gradient-based method to extract temporal and spectral relevance map from raw waveform inputs, and improve the understanding of raw waveform-based CNNs.

Belinkov and Glass [2017] analyse hidden representations in an end-to-end acoustics-to-characters ASR model. They evaluate the representations at different layers and compare their quality for predicting phone labels, to evaluate if the model implicitly learns phonetic representations. Our work in Section 6.4 is similar to this work in a sense that we also evaluate the intermediate representations to learn what the model has learned implicitly, however, we measure the performance of speaker and acoustic conditions prediction performance, rather that phone prediction.

A similar methodology is used by Elloumi et al. [2018], but the analysis of learned representations is used to predict ASR performance for unseen broadcast programs. The authors show that the hidden layers convey information about speech style, accent and broadcast type.

Another application example of methods used to better understand the representations in speech recognition is data selection. Mitra and Franco [2018] propose to analyse the output layer activations to perform data selection for unsupervised AM adaptation, by measuring the distance between two most likely targets and using it as a measure of goodness of the hypothesis.

6.2 Simplifying deep CNN architectures

This section is based on [Rownicka et al., 2017], where we simplify deep CNN models, to discover the performance contribution from different components. We propose a number of simplified deep CNN architectures, taking into account the use of fully-connected layers and down-sampling approaches. We investigate three ways to down-sample feature maps: max-pooling, average-pooling, and convolution with increased stride. By using controlled architecture design, with the empirical comparisons of the results, we shed some light on the most beneficial components in a CNN-based acoustic model. For the experiments, we use Aurora-4 and MGB-3 datasets, in order to evaluate the simplified architectures in challenging, noisy acoustic conditions.

Most deep CNN acoustic models have used architectures in which the upper hidden layers are fully-connected [Sercu et al., 2016a; Sercu and Goel, 2016a; Qian and Woodland, 2017; Tan et al., 2018]; in contrast, Yu et al. [2016a] use an architecture without any fully-connected layers. Furthermore, they also do not employ pooling layers – convolutional layers with a larger stride are used to down-sample the feature maps. However, Yu et al. [2016a] compare the resultant acoustic model architecture to DNN and long short-term memory (LSTM) recurrent neural network (RNN) models, with no comparison across CNN architectures, which makes it difficult to assess the contribution of the CNN architectural components – it is not clear whether removing fully-connected layers and replacing pooling layers with convolutional layers improves or degrades the accuracy of the speech recognition system.

In this work we investigate deep CNNs to find out which components are necessary to achieve the state-of the art accuracy for robust speech recognition. Specifically, we want to learn the importance of different components and strategies for the network (fully-connected layers, pooling layers, convolution stride) in noisy speech recognition.

6.2.1 Experiments

We use a hybrid NN-HMM speech recognition system. A Gaussian Mixture Model (GMM) based HMM system is first trained to estimate the context-dependent phone models, from which a forced alignments is obtained which provides the targets for training all our NN-HMM systems. We use Kaldi [Povey et al., 2011]

to train the GMM-HMM system with ∼3.4k (for Aurora-4) or ∼9.4k (for MGB) clustered states using maximum likelihood estimation criterion and MFCC features transformed with LDA, MLLT, and fMLLR transforms.

In all of our experiments with neural network acoustic models we used FBANK features as inputs. Following the findings in [Qian and Woodland, 2017], where using only static features was the best setup for robust speech recognition, we also use only one feature map at the input. We use 40 filters and 5 frames of context, so the input feature map size is 11×40.

We used the TensorFlow software framework [Abadi et al., 2015] to train the NN models and the tfkaldi software [Renkens, 2016] to integrate TensorFlow neural networks with Kaldi.

Figure 6.1 shows the architectures of the convolutional models analysed in this work. For the hidden units, all the networks used a rectifier linear unit (ReLU) non-linear activation function. The baseline DNN model comprised 6 hidden layers with 2048 nodes each. The baseline CNN model followed Sainath et al. [2015b] and comprised two convolutional layers with kernel sizes 9×9 and 3×4 respectively. There are 256 feature maps in the first convolutional layer and 512 in the second one. In the baseline CNN, model we use overlapping pooling and valid convolutions (no padding). In the first convolutional layer, the kernels were applied using a stride of 1. In the second convolutional layer, the kernel's strides were $s_{time} = 1$ and $s_{freq} = 4$.

For all deep CNN models we use non-overlapping pooling, together with zero-padding at the output of each convolutional layer to ensure feature map size consistency within each CNN block. A CNN block is a stack of convolutional layers with kernel stride size $s_{time} = 1$ and $s_{freq} = 1$ (stride 1×1). In all our models we use 5 CNN blocks with 2-3 convolutional layers in each block. CNN blocks were alternated with down-sampling layers: max pooling, average pooling, convolutional layers reducing the feature maps size only in frequency dimension (stride 1×2), or convolutional layers reducing the feature maps size in both time and frequency (stride 2×2).

Our baseline deep CNN model *deep CNN-max-4FC* used max-pooling; the 2D output of the last pooling layer was transformed into a vector and fed into three FC hidden layers before the softmax output layer. The remaining deep CNN architectures that we investigated did not use any FC hidden layers after the final down-sampling layer. In *deep CNN-avg* the max-pooling layers were replaced with

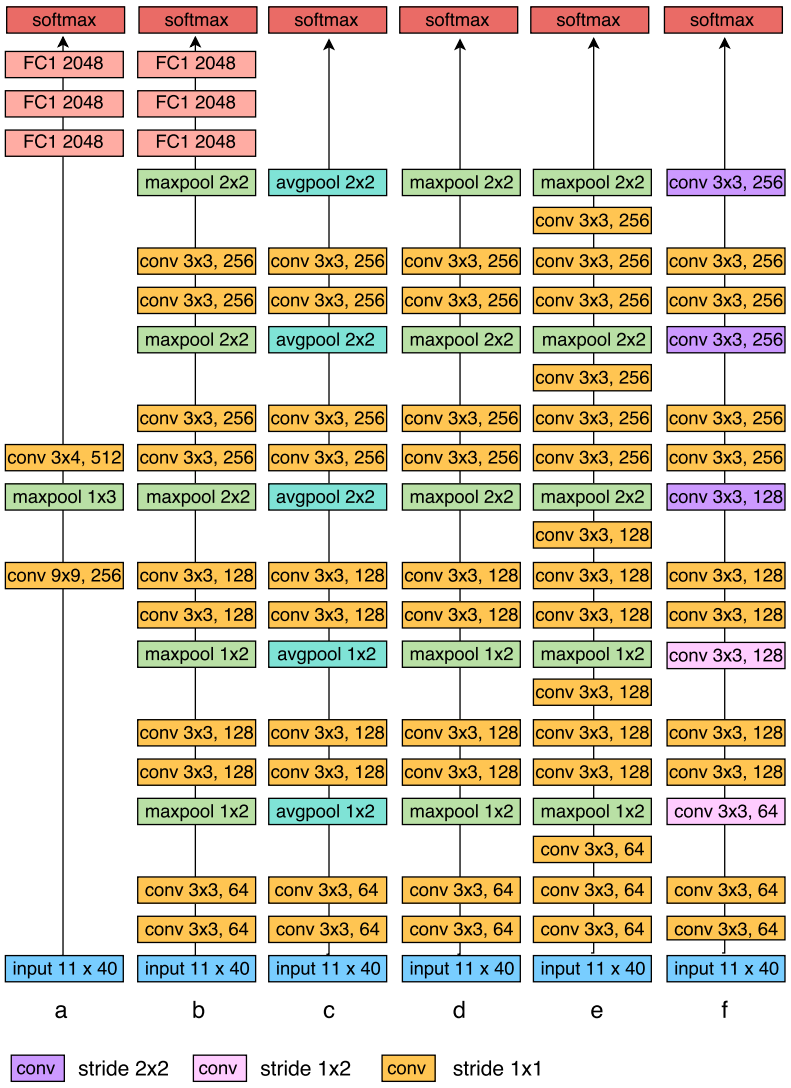


Figure 6.1: Architectures of the convolutional models: a) *CNN*, b) *deep CNN-max-4FC*, c) *deep CNN-avg*, d) *deep CNN-max*, e) *deep CNN-max-addconv*, f) *deep CNN-allconv*.

average pooling using the same kernel sizes and strides as in the baseline deep CNN model; the output of the last pooling layer is fed directly into the softmax output layer. *deep CNN-max* differed from *deep CNN-avg* by using max-pooling in place of average pooling, whereas *deep CNN-allconv* used convolutional layers instead of pooling layers to perform down-sampling[3]. The strides of the kernels in the down-sampling convolutional layers matched the strides of the pooling layer kernels in our other deep CNN models, and the number of output channels of down-sampling convolutional layers matched the number of their input channels.

Introducing down-sampling convolutional layers into the model resulted in an

[3]This model architecture is equivalent to the baseline CNN in Chapter 5 (Table 5.1).

Model	# params	$t_{Aurora4}$	t_{MGB-3}
DNN	24.7M	8 h	57 h
CNN	23.8M	–	–
deep CNN-max-4FC	20.1M	17 h	155 h
deep CNN-avg	6.1M	55 h	147 h
deep CNN-max	6.1M	16 h	131 h
deep CNN-max-addconv	7.6M	31 h	–
deep CNN-allconv	7.6M	20 h	215 h

Table 6.1: Number of parameters (# params) and training times for Aurora-4 ($t_{Aurora4}$) and MGB-3 (t_{MGB-3}) datasets for the baseline models (DNN, CNN, deep CNN-max-4FC) and our deep CNN models (avg, max, max-addconv, allconv).

increase of the overall number of parameters. Therefore, to compare networks of similar complexity, the *deep CNN-max-addconv* model contains an additional convolutional layer in each convolutional block. The numbers of weights for each model are summarized in Table 6.1. By removing FC layers from the network architectures, our proposed deep CNN models greatly reduce the number of the independent parameters of the networks. In Table 6.1 we also show the training times for the models trained on the same GPU (Nvidia TITAN X).

All models were trained using the cross-entropy criterion, optimised with mini-batch stochastic gradient descent using the Adam algorithm to smooth the gradient estimates. To keep the scale of the gradients and activations roughly the same in all layers we used "Xavier" initialization [Glorot and Bengio, 2010] for the weights. We seeded the random number generator to a nonnegative integer in all our experiments. The initial learning rates $\gamma \in \{0.002, 0.001, 0.0008, 0.0005\}$ were individually adapted for each model. The distribution of the outputs of each layer was normalised using batch normalisation [Ioffe and Szegedy, 2015]. The training termination is conditioned on the development loss.

6.2.2 Results and discussions

We evaluated the acoustic models on the Aurora-4 task and on the much larger MGB tasks – the MGB-3 training set is over 20 times bigger than the Aurora-4 training set.

Model	A	B	C	D	AVG
DNN	3.47	7.67	7.85	19.73	12.55
CNN	3.33	6.89	6.59	17.92	11.34
deep CNN-max-4FC	2.43	5.92	5.74	16.26	10.09
deep CNN-avg	2.75	5.88	7.27	16.46	10.29
deep CNN-max	2.56	5.78	5.36	15.40	9.64
deep CNN-max-addconv	2.50	6.02	6.95	16.35	10.26
deep CNN-allconv	2.32	5.45	5.38	15.56	**9.55**

Table 6.2: Word Error Rates (WERs) [%] for Aurora-4 test sets A, B, C, D, and the average (AVG) WER for the baseline models (DNN, CNN, deep CNN-max-4FC) and our deep CNN models (avg, max, max-addconv, allconv) trained with alignments generated from multi-condition training set of Aurora-4.

Evaluation on Aurora-4

We trained Aurora-4 models using two different alignments. The results in Table 6.2 were obtained with alignments produced by a GMM-HMM systems trained using the multi-condition Aurora-4 training set. We prefer this approach as it better matches the usual condition in which synchronized clean-condition training sets are not available for a task. However, in order to compare our results with other Aurora-4 results in the literature (which often use clean-condition training to generate the alignments for the NN-HMM systems), Table 6.3 compares some of our multi-condition alignment results with the results using clean-condition alignments for NN training.

Our baseline models – *DNN, CNN, deep CNN-max-4FC* – achieve lower WERs than similar models in the literature. For instance, the best deep CNN model in [Qian et al., 2016] achieves a WER of 8.81% (with clean-condition alignments) compared to our 8.70% for the same network architecture. However, introducing residual connection into a VGG-like architecture resulted in a lower WER of 8.10%, and using factor-aware training reduced the WER down to 7.65% [Tan et al., 2018]. The state-of-the-art WER for Aurora-4 reported in [Tan et al., 2018] is 5.67%; it was achieved with a multi-pass ASR system which combines several adaptation approaches.

Our DNN baseline gives to our knowledge the lowest DNN WER on Aurora-4 (10.79% compared to 11.11% in [Qian and Woodland, 2017; Qian et al., 2016; Tan

et al., 2018]). These improvements in accuracy may result from differences in our training procedure – using the Adam optimizer (rather than stochastic gradient descent with the NewBOB learning rate schedule) and batch normalisation. Our best deep CNN model, without any additional input features nor additional neural network models, achieves a WER of 7.75%.

We consider the four simplified deep CNN architectures for robust speech recognition trained using the more realistic targets obtained from multi-condition alignments (Table 6.2). These results indicate that removing the fully-connected layers from the deep CNN model (*deep CNN-max*) improves the accuracy for test subsets B, C, and D. The gain is the most prominent for the most corrupted test subset D, which indicates that this kind of architecture may be beneficial also for other datasets corrupted with noise and with mismatched channels. Removing the majority of the independent parameters of the network resulted in WER reduction over the whole test set.

However, the *deep CNN-avg* model slightly outperforms the baseline deep CNN only for the test subset B; the overall WER for this model is higher than the baseline WER. This model performs the worst for subsets A and C. This may be explained by the smoothing properties of the average pooling layer which for relatively clean test subsets may cause the performance degradation.

Among all models trained with multi-condition alignments, the average WER over test subsets A, B, C, and D is the lowest for the *deep CNN-allconv* model, consisting solely of convolutional layers and a softmax layer, and without any fully-connected hidden layers. The relative improvement over the corresponding baseline deep CNN model is 5%, with a 16% relative improvement over the baseline CNN, and 24% over the baseline DNN[4]. The largest relative WER reduction was over subset B (additive noise) compared to the deep CNN baseline, making the proposed *deep CNN-allconv* architecture a promising choice for other noisy speech recognition tasks.

By aggressively reducing the number of parameters of a network by removing all FC layers and by introducing five additional convolutional layers to the deep CNN model, the capacity of the model is increased without considerably increasing the amount of the model's weights. This may enable the model to perform

[4]The results for a baseline CNN for Aurora-4 in Chapter 5 differ because of a different batch normalisation implementation. In Chapter 5, at test time we normalised the output of a layer using moving mean and variance instead of using the statistics computed over the complete training set.

Model	A	B	C	D	AVG
DNN/mc	3.47	7.67	7.85	19.73	12.55
DNN/cln	3.19	6.42	7.04	17.04	**10.79**
deep CNN-max-4FC/mc	2.43	5.92	5.74	16.26	10.09
deep CNN-max-4FC/cln	2.54	5.33	4.61	13.77	**8.70**
deep CNN-allconv/mc	2.32	5.45	5.38	15.56	9.55
deep CNN-allconv/cln	2.43	4.43	4.50	12.50	**7.75**

Table 6.3: Word Error Rates (WERs) [%] for Aurora-4 test sets A, B, C, D, and the average (AVG) WER for models trained with alignments generated from multi-condition training set (mc) and synchronized clean-condition training set (cln) of Aurora-4.

better generalizations in training and as a result perform better on disrupted test sets. The final proposed model, *deep CNN-max-addconv*, is worse than the deep CNN baseline for all test subsets, although this experiment indicates that the performance gain for *deep CNN-allconv* model is not solely due to model size expansion.

For the clean-condition alignments, the relative improvement for most accurate model is 11% over the corresponding deep CNN baseline, and 28% over the DNN baseline (Table 6.3). These results indicate that the proposed *deep CNN-allconv* architecture benefits the most from the clean alignments setup. For clean alignments, the relative performance gains over our deep CNN baseline for test subsets A, B, C, and D are 4%, 17%, 2%, and 9%, respectively. The test subset B (with additive noise) and subset D (with additive noise and microphone mismatch) are again the ones benefiting the most from the proposed deep CNN architecture. For the same models architectures, the relative improvements for clean-aligned over multi-aligned models are 19% for *deep CNN-allconv*, and 14% for the baseline *deep CNN-max-4FC*. The choice of the alignments therefore has a major influence on the final WERs in the Aurora-4 task.

Evaluation on MGB

To ensure the reliability of our results, and to scale to a larger, more realistic task, we evaluated our models on the MGB Challenge data. We used the MGB-3 training set to train our acoustic models. To provide calibration for the acoustic and language models used in the MGB task, we evaluate the models trained on

Model	**WER**	
	MGB3-dev	MGB1-test
deep CNN-max-4FC	53.2	42.4
deep CNN-avg	52.4	41.4
deep CNN-max	52.5	41.4
deep CNN-allconv	**52.2**	**41.2**
deep CNN-allconv-4G	**50.0**	**38.7**

Table 6.4: Word Error Rates [%] for MGB-3 dev set and MGB-1 test set for the baseline model (deep CNN-max-4FC) and our deep CNN models (avg, max, allconv). The last deep CNN-allconv-4G model is re-scored with a 4-gram LM. All models are trained on MGB-3 training data.

MGB-3 training data on both the MGB-3 development set and on the MGB-1 test set. It should be noted, however, that our MGB-1 results can not be directly compared to the other results in the literature for MGB-1 task as the training set used in this work is MGB-3 which contains about 3.5× less audio data than the MGB-1 set, drawn from a different set of multi-genre TV programmes.

WERs for the MGB-3 dev set and the MGB-1 test set are summarised in Table 6.4. In those tasks we did not aim for best possible performance; our objective was to compare the proposed deep CNN models with a deep CNN baseline and with each other on a task larger than Aurora-4[5]. For both MGB test sets we obtained the lowest WERs using our proposed *deep CNN-allconv* model which is in line with the Aurora-4 findings above. It also confirms that this kind of architecture can be beneficial for other robust speech recognition tasks, with small but consistent gains over the deep CNN – 1.0% WER absolute improvement for MGB-3 and 1.2% for MGB-1. The WERs for average and max pooling are comparable in both MGB tasks. We also show the results of re-scoring our best acoustic model with a 4-gram language model, giving an additional 2.2% and 2.5% absolute improvement for MGB-3 and MGB-1 tasks, respectively.

[5]For instance, further reductions in WER would be obtained through sequence training, speaker adaptation, improved test set segmentation, and re-alignment.

6.2.3 Conclusions

In this section, we investigated deep CNN architectures for robust speech recognition, exploring the effect of using different components of the architecture on the performance. We explored several simplified architectures to investigate which deep CNN components are necessary to obtain state-of-the-art results for robust speech recognition task. We investigated different down-sampling strategies – our experiments demonstrate that max-pooling performs better than average pooling for smaller, Aurora-4 database, but pooling layers are not necessary at all to achieve state-of-the-art results for speech recognition with deep CNNs. Instead, using convolutional layers with increased stride can effectively enable robust representation learning. Convolution with increased stride performs convolution and down-sampling at the same time, while pooling is a fixed down-sampling operation. We hypothesise that the additional parameters and the down-sampling ability contribute to the performance gains. Moreover, using a convolutional layer with increased stride reduces the amount of computations, compared to a stack of a convolutional layer with stride $s = 1$ and a pooling layer. The performance gains resulting from the choice of the down-sampling approach vary depending on the dataset and the alignments used.

In addition, removing fully-connected layers from a deep CNN architecture, typically used in speech recognition, contributed most to the performance gains in our experiments, especially for noisy test data. We hypothesise that removing the upper fully-connected layers prevented the model from overfitting. Our model consisting solely of fifteen 2D convolutional layers with the same kernels sizes throughout the network and a single softmax classification layer gave the best performance consistently in our experiments on the Aurora-4 and MGB tasks.

6.3 Visualisations

We explore two approaches to visualisation as a way to better understand learned representations. First, we explore local explanations (Section 6.3.1). Secondly, we attempt to visualise the concepts learned by the acoustic models (Section 6.3.2), as a way to approximate distances in the original high-dimensional decision space, and provide new interpretable information.

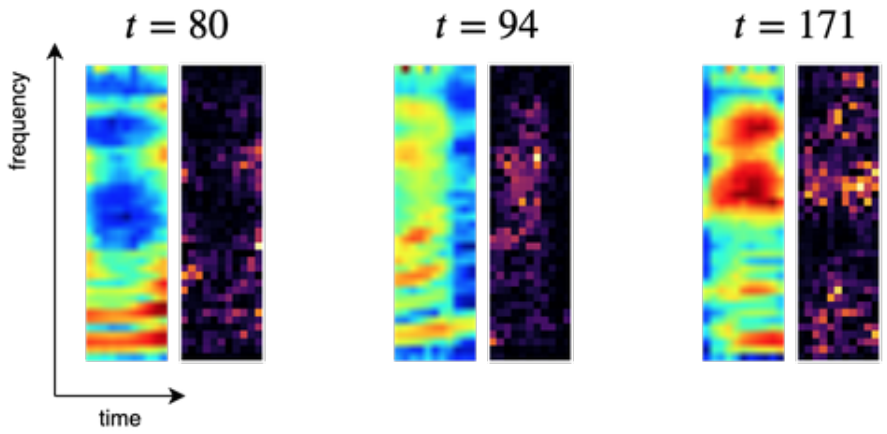


Figure 6.2: Input feature maps (left) and corresponding saliency maps (right) for selected time steps t. Time dimension (11 time frames) is shown horizontally, frequency (40 FBANKs) – vertically.

6.3.1 Explaining individual predictions

In the previous section we showed that for the Aurora-4 task, the lowest WER was achieved by a fully-convolutional model with fifteen layers (*deep CNN-allconv*). We therefore analyse the activations from this model. We generate saliency maps with a single back-propagation pass for input time-frequency feature maps $X_t \in \mathbb{R}^{h \times w}$ at time t. We compute the gradients of the probability of class c with respect to the feature map X_t as

$$g_c(t) = \frac{\partial L_c}{\partial X_t} \tag{6.1}$$

where L_c is class c logit (unnormalised prediction, i.e. input to the softmax layer), and class c is the predicted class (i.e. PDF identifier) for input X_t. The locations where the derivatives take high values indicate features important for the classification, as changing them would change the prediction. To obtain a saliency map, we discard the sign of the gradients.

We use Picasso visualisation toolkit [Henderson and Rothe, 2017] as the interface to explore the visualisations with our CNN acoustic model. In the figures below, we show the saliency maps at time t for a single test utterance from Aurora-4 test set A (utterance ID is 441c02130) and the corresponding input feature maps.

In Figure 6.2 we can observe that the locations of the high-value saliency features roughly correspond to the locations of the input features of high magnitude.

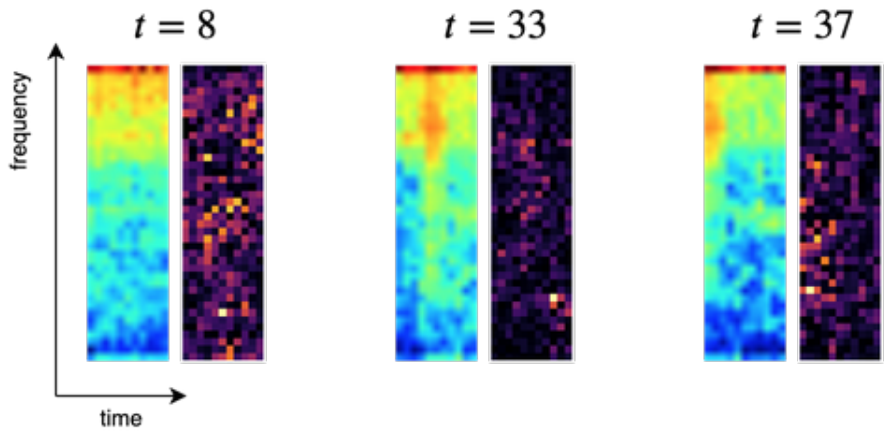
This result is a confirmation of how the model interprets the input – *it encodes the location of the FBANK features the most important for the classification.* We find this observation interesting as it confirms that the deep CNN acoustic model is capable of learning non-local relationships between frequency bands. We hypothesise that the reason for this behaviour is stacking the convolutional layers with small kernels (3×3 in this work) on top of each other, which increases the effective receptive field, and enables the model to learn non-local patterns. Due to padding and increased stride, the effective receptive field spans over all forty frequency dimensions already at layer nine. Therefore, the model can learn the relationships between all frequency bands, which is confirmed by the saliency analysis.

Saliency maps can therefore be used to better understand a highly non-linear decision space by showing the regions of attribution in the input. However, for speech recognition, those visualisations do not provide a lot of new information. One possible application of the saliency maps could be frame-level speech detection and segmentation. We evaluate this idea by looking at saliency maps for non-speech frames.

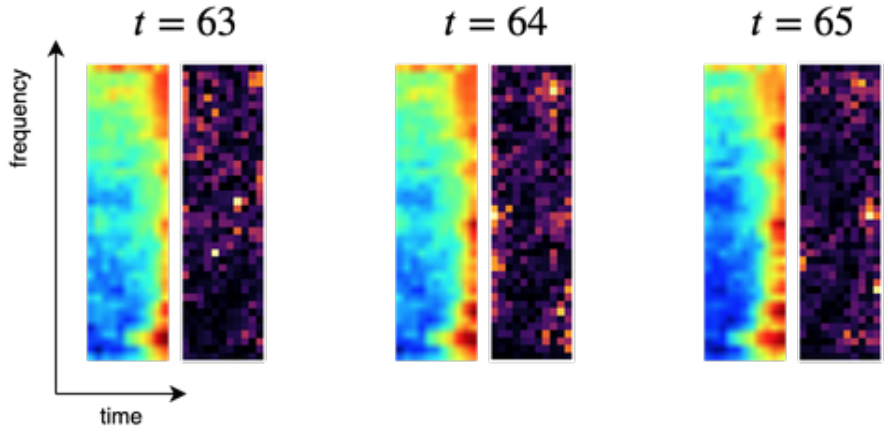
Figure 6.3 shows saliency maps for initial time frames (classified as silent) from the same utterance. Just by looking at the saliency maps, it is hard to determine if the input feature map corresponds to silence or speech. We also look at the frames where the speech starts (i.e. first three frames classified as non-silence), to see if the saliency maps could provide information useful for speech activity detection. Here, it is also hard to identify the speech regions in the input feature maps, with just the visual analysis of the saliency maps. Furthermore, we observe that the saliency maps of inputs shifted only by one frame differ between each other a lot (Figure 6.3b), making those visualisations hard to interpret, even within a single utterance. Therefore, this local interpretability method can be used to investigate specific predictions within a short time frame, but is not very informative for the analysis of a sequential signal.

6.3.2 Concepts learned by the model

We next turn to t-SNE for more global visualisations. We are interested if this approach can be used to better understand decisions made by the



(a)



(b)

Figure 6.3: Input feature maps (left) and corresponding saliency maps (right) for selected time steps t. (a) - frames classified as silent, (b) - first three frames of utterance 441c02130 classified as non-silence.

acoustic model. We explore several perplexities[6] for the visualisations ($p = 30, 40, 50, 100, 330, 600, 2300$), and decide to use 40 for all images in this section (we did not observe major differences in the quality of visualisations with different values). We use scikit-learn [Pedregosa et al., 2011] implementation of t-SNE with Barnes-Hut approximation used for optimisation [van der Maaten, 2013].

We extract the Acoustic Model utterance embeddings as described in Section 4.3.1. For reference, Figure 6.4 shows the flowchart of the embedding extraction for the test set. Acoustic Model (AM) embeddings are utterance summaries

[6]It is a modelling hyper-parameter determining the attention to the local and global aspects in the data.

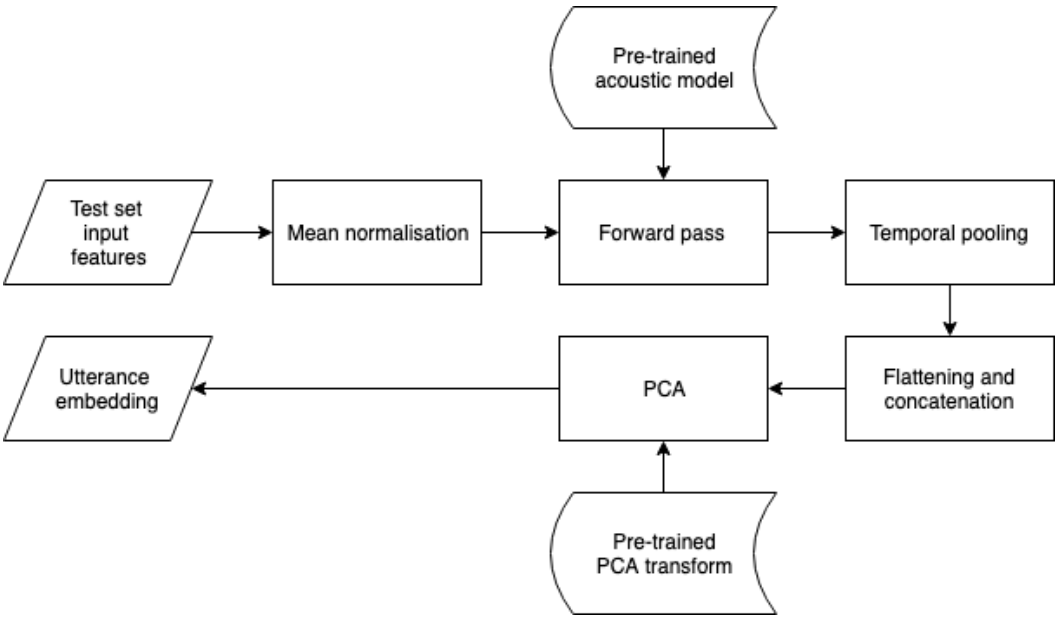
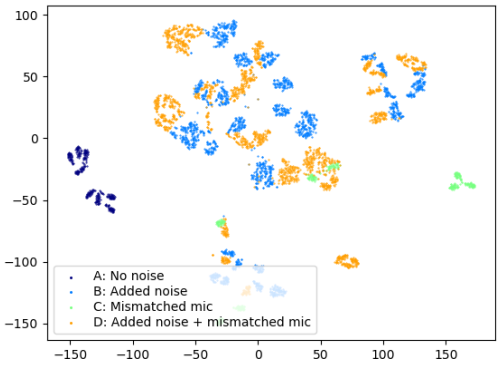


Figure 6.4: Flowchart of the Acoustic Model (AM) utterance embedding extraction process for test utterances.

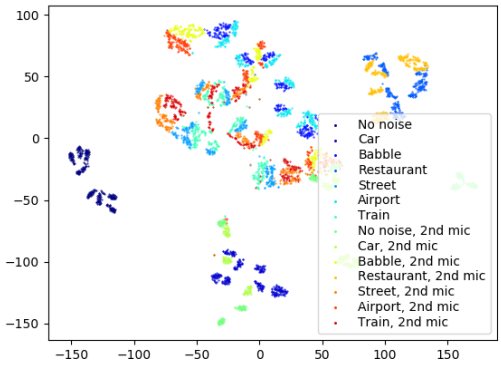
derived from a DNN or deep CNN acoustic model, trained to output posterior probabilities of senones. We extract utterance-level information with the use of temporal pooling, and we concatenate the representations from different layers into a single vector. The PCA transform is used to select the directions of the highest variance for the final utterance embedding. The pre-trained acoustic model and PCA transform are trained using multi-condition Aurora-4 training set. The mean normalisation is speaker-wise. We then embed the utterance embeddings into a low-dimensional explanation space with t-SNE algorithm. We follow this procedure for the whole test set of Aurora-4, with the aim to give a global view of the dataset.

Figure 6.5 shows the utterances organised with t-SNE by their CNN activation values. We use different categories for labelling (test subset, acoustic condition, noise type, microphone, speaker, gender). For comparison, Figure 6.6 shows utterance embeddings extracted from a DNN model.

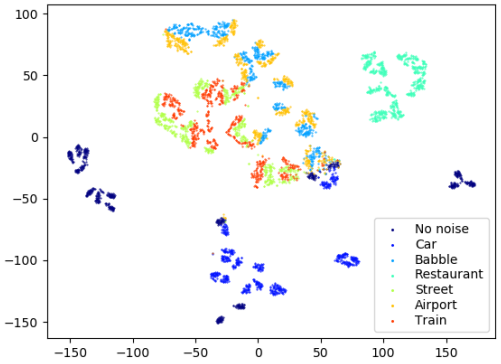
Both CNN and DNN embeddings seem to be organised by different attributes of the utterances. The utterances belonging to the same class are close to each other in this two-dimensional explanation space, also across different categories. For example, in Figure 6.5(c) there is a cluster of utterances with “no noise" label on the far left side of the figure. At the far right there is a “no noise" cluster too. By looking at Figure 6.5(d) we find an explanation – the far right cluster



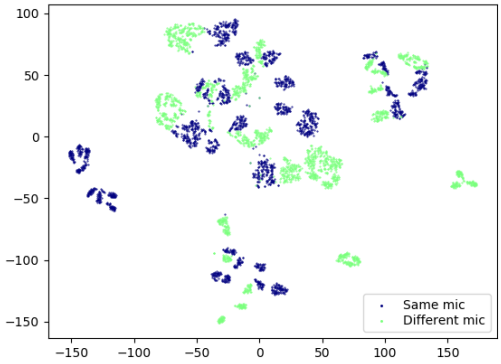
(a) Test subset



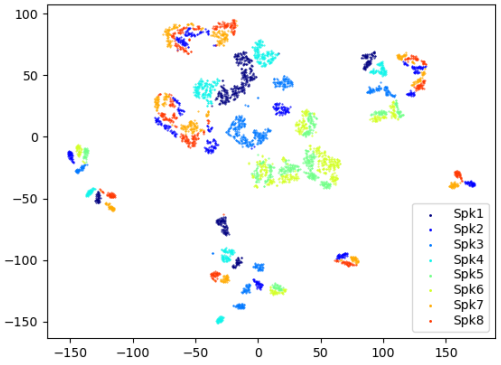
(b) Acoustic condition



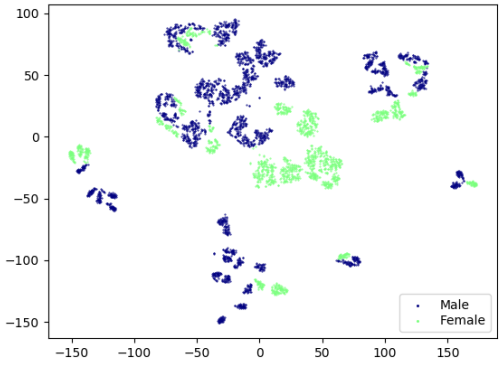
(c) Noise type



(d) Microphone

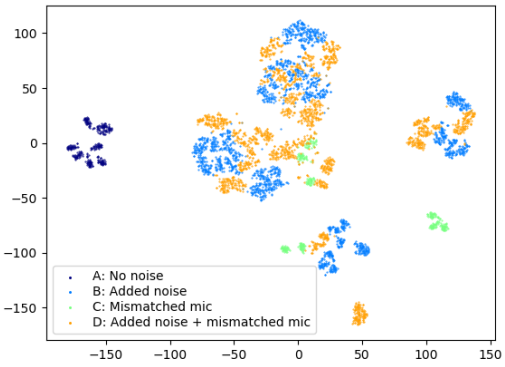


(e) Speaker



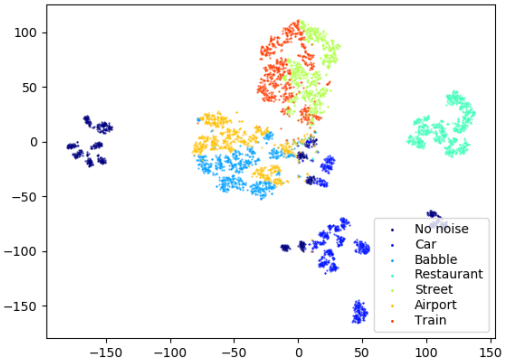
(f) Gender

Figure 6.5: t-SNE of utterance embeddings for Aurora-4 test utterances generated with a deep CNN model labelled according to a category indicated in the caption.

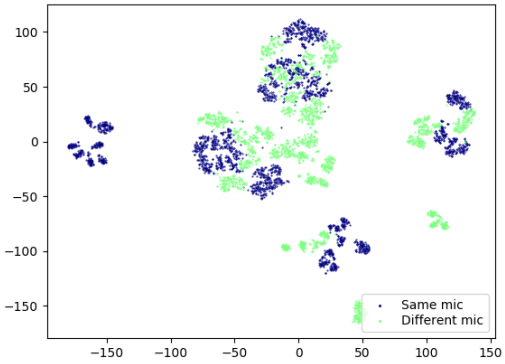


(a) Test subset

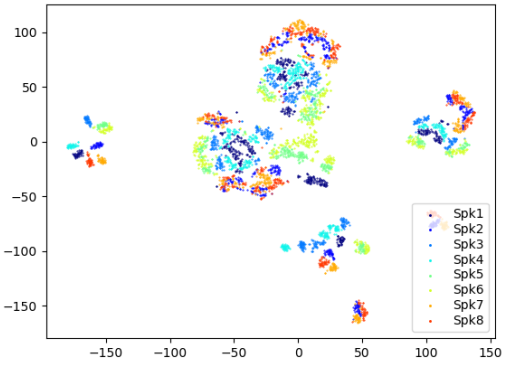
(b) Acoustic condition



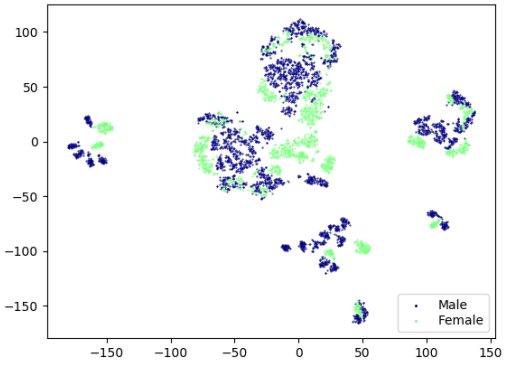
(c) Noise type



(d) Microphone



(e) Speaker



(f) Gender

Figure 6.6: t-SNE of utterance embeddings for Aurora-4 test utterances generated with a DNN model labelled according to a category indicated in the caption.

of utterances was recorded with a different microphone. Moreover, Figure 6.5(e) provides an explanation for high-density clusters – the utterances were produced by different speakers.

We find those visualisations interesting, especially because the exact same attributes were not seen during training (e.g. speakers do not overlap in training and testing), and yet the two investigated models seem to respond to the features corresponding to test attributes (speaker identity). We infer that the investigated models have robustly learned about attributes such as speaker identity, noise and microphone type.

Comparing CNN and DNN embeddings, we can see that CNN representations are more densely clustered. Perhaps this can be explained by more local correlations learned by a CNN model compared to a DNN, due to local receptive fields and small 3×3 kernels used throughout the network. The advantage of the DNN seem to be in better separability of the clusters corresponding to the noise type (Figure 6.6(c)). However, since the WER was lower for the CNN model than for the DNN model[7], it might be more beneficial for noise robust speech recognition to learn about local noise correlations (as shown in Figure 6.5(c)) than about the true underlying noise type (as shown in Figure 6.6(c)).

Another difference between CNN and DNN plots can be observed by looking at speaker clusters (Figures (e)). For the CNN, although speakers are dispersed, there are a couple of dense clusters per speaker. On the other hand, for the DNN, the clusters are much less dense with less obvious boundaries between speaker clusters. We hypothesise that the dense CNN clusters are again caused by locality imposed in the architecture. As for the DNN, the information about the speakers in the activations seems to be less distinguishable than in the CNN activations.

The analysis demonstrates that t-SNE can be used to gain some global insight into how the models are processing the data. We organised the utterances by their activation values and we used the metadata provided in the Aurora-4 task to show which utterances are seen as similar by the model. This analysis helps to infer some ideas about what features the network is responding to. We have observed higher density of the CNN clusters compared DNN clusters, which is in line with the assumptions of the architectures. We believe that this can be one of the factors contributing to the superior performance of the CNN model for the

[7] 9.55% for the CNN compared to 12.55% for the DNN.

Aurora-4 task.

In the next section we strive to quantify the properties of the activations, to validate the observations from this section.

6.4 Diagnostic verification

We evaluate utterance-level DNN and CNN embeddings in secondary verification tasks to quantify if latent representations contain the information about different factors of variation, such as speaker identity[8]. We do not consider senone recognition in this section because we are interested in implicit learning of the underlying factors of variation, other than those directly related to the pronunciation of different sounds. Implicit learning would indicate the relevance of the underlying factors for speech recognition, and would suggest that a robust representation should be able to disentangle this factor.

For Aurora-4 we perform speaker, gender, noise and acoustic condition recognition and compare our embeddings with i-vectors (Section 6.4.1). Furthermore, we study the encoding properties of node activations in various layers of the network, to learn how the attributes are encoded across layers. For AMI we look at speaker recognition and extend the comparisons to x-vectors (Section 6.4.2). Finally, we evaluate the CNN embeddings for speaker recognition for the VoxCeleb corpus (Section 6.4.3).

We do not use any non-linear multi-layer classifiers in our experiments because our goal is to better understand learned representations – we want to make sure that the results are not due to a powerful classifier but that they reflect the properties of the embeddings.

We also perform verification rather than identification. Equal Error Rate (EER), where false rejection rate equals to false acceptance rate, is a common measure of performance for Automatic Speaker Verification (ASV) systems. We also use it as a primary evaluation metric in this section. It reflects the trade-off between false rejection and false acceptance rates, where

$$\textit{False rejection rate} = \frac{\textit{Number of false rejection}}{\textit{Number of legitimate attempts}}$$

[8]In NLP, a similar evaluation technique is referred to as *probing* [Tenney et al., 2019].

$$\textit{False acceptance rate} = \frac{\textit{Number of false acceptance}}{\textit{Number of imposter attempts}}$$

Each trial for scoring is an enrolment/evaluation pair – the evaluation embeddings $\boldsymbol{x}_{eval}$ are compared with the enrolment embeddings $\boldsymbol{x}_{enrol}$, i.e. mean vectors across speakers, to obtain a score. Then, the trial is accepted or rejected based on a score threshold.

In our experiments we use three scores for evaluation: cosine distance score s_{cos}, Linear Discriminant Analysis (LDA) score s_{LDA}, and Probabilistic Linear Discriminant Analysis (PLDA) score s_{PLDA}.

Cosine distance score (cosine back-end) is a normalised dot product of enrolment and evaluation vectors, $\boldsymbol{x}_{enrol}$ and $\boldsymbol{x}_{eval}$. We first subtract the global mean computed over N embeddings in the training set $\boldsymbol{x}_{train}$, as $\bar{\boldsymbol{x}} = \frac{\sum_{i=1}^{N} \boldsymbol{x}_{train,i}}{N}$, to obtain mean-normalised vectors $\hat{\boldsymbol{x}}_{enrol}$ and $\hat{\boldsymbol{x}}_{eval}$. We then normalise them to unit length by dividing by the norm of the vectors, and compute the dot product to obtain the cosine score:

$$s_{cos}(\boldsymbol{x}_{enrol}, \boldsymbol{x}_{eval}) = \frac{\hat{\boldsymbol{x}}_{enrol}}{||\hat{\boldsymbol{x}}_{enrol}||} \cdot \frac{\hat{\boldsymbol{x}}_{eval}}{||\hat{\boldsymbol{x}}_{eval}||}. \tag{6.2}$$

For the LDA score (LDA back-end), we first compute the LDA transformation matrix $\boldsymbol{T}(.)$, supervised by the training labels corresponding to speakers (for speaker verification), and use it to transform enrolment and evaluation embeddings, before length normalisation and dot product computation:

$$s_{LDA}(\boldsymbol{x}_{enrol}, \boldsymbol{x}_{eval}) = \frac{\boldsymbol{T}(\boldsymbol{x}_{enrol})}{||\boldsymbol{T}(\boldsymbol{x}_{enrol})||} \cdot \frac{\boldsymbol{T}(\boldsymbol{x}_{eval})}{||\boldsymbol{T}(\boldsymbol{x}_{eval})||}. \tag{6.3}$$

For the PLDA score (LDA/PLDA back-end), we compute the PLDA transformation matrix on the training embeddings, after reducing the dimensionality with the LDA transform $\boldsymbol{T}(.)$. PLDA model is then used to compute the log likelihood ratio:

$$s_{PLDA}(\boldsymbol{x}_{enrol}, \boldsymbol{x}_{eval}) = \log \frac{p(\boldsymbol{T}(\boldsymbol{x}_{enrol}), \boldsymbol{T}(\boldsymbol{x}_{eval})|H_1)}{p(\boldsymbol{T}(\boldsymbol{x}_{enrol})|H_0)p(\boldsymbol{T}(\boldsymbol{x}_{eval})|H_0)}, \tag{6.4}$$

where H_1 is the hypothesis that the enrolment and evaluation speakers are the same, and H_0 – that they are different.

6.4.1 Aurora-4

We use standard Aurora-4 test set to create two disjoint sets (evaluation and enrolment) for the experiments in this section. Both sets consist of 2310 utterances chosen randomly from Aurora-4 test set, such that all 8 speakers are present in both sets and the number of utterances per speaker are balanced across both sets. The type of the acoustic conditions in the training set match the conditions in enrol and eval datasets. Speakers in the training set do not overlap with the enrolment (enrol) and evaluation (eval) datasets. The multi-condition training set is used for NN, PCA, LDA and PLDA training. The whole model AM enrolment and evaluation embeddings are extracted according to the flowchart in Figure 6.4, with speaker-wise mean normalisation.

The target/nontarget proportion for the trials used for scoring is 50%, which results in 4620 trials. Each evaluation vector at the utterance level is scored against the enrolment vectors averaged for speakers, acoustic conditions, noise or gender, depending on the task.

Whole model AM embeddings are designed to capture all of the information learned by the NN models by projecting to a common feature space. We evaluate them for speaker, acoustic condition, noise, and gender recognition and we compare them to i-vectors, which are commonly used to characterise utterances. These evaluations aim to demonstrate the differences between different utterance-level representations in terms of the amount of interpretable information that they contain, to verify the visualisations in the previous section.

We reduce the dimensionality of the whole model embeddings to 400 with the PCA transform; the dimensionality of 400 was chosen such that the amount of variance to be explained by all of the selected components (of the training set) is greater than 99.9%. Table 6.5 shows the distribution of the origin of highest variance components by layers for the DNN and the deep CNN whole model acoustic embeddings.

The PCA components in the deep CNN embedding are more evenly distributed among layers, whereas the majority of the highest variance PCA components in the DNN embedding come from the last fully-connected layer. This result indicates that the representation learning mechanism is different between the two models and that the models may differently learn about speech attributes. Further experiments aim to investigate those differences in more detail.

DNN embeddings		deep CNN embeddings	
FC0	3.25%	*conv1*	13.50%
FC1	3.50%	*conv2*	13.75%
FC2	1.50%	*conv3*	13.25%
FC3	0.50%	*conv4*	22.75%
FC4	5.00%	*conv5*	36.75%
FC5	86.25%		

Table 6.5: Distribution of the origin of the highest variance PCA components by layers for 400-dimensional DNN and deep CNN whole model embeddings.

Speaker recognition

We extract i-vectors on FBANK features to match the features used in the AM utterance embedding extraction, as well as on mel-frequency cepstral coefficient (MFCC) features to obtain better quality i-vectors for fair comparison, in both cases using a full-covariance Universal Background Model (UBM) with 2048 components. The i-vector dimensionality matches the dimensionality of the AM embeddings (400), and they are extracted per utterance.

Table 6.6 shows the results of applying the extracted embeddings, as well as i-vectors, to the speaker recognition task. For the LDA and LDA/PLDA back-ends we use gender independent models. In the LDA back-end, the dimensionality was 10 for all vector types. In the LDA/PLDA back-end LDA is used to decrease the dimensionality prior to PLDA; this was optimised for each vector type separately (50 for FBANK i-vectors, 70 for MFCC i-vectors, 55 for DNN embeddings, and 70 for deep CNN embeddings). Both i-vectors and our proposed DNN and deep CNN embeddings are computed per utterance and are length normalised.

i-vectors computed on MFCC features give the best performance when evaluated with a cosine similarity back-end. Here, all of the dimensions of the vectors being scored are used. The i-vectors computed on MFCC features contain more information about the speakers compared to raw deep embeddings. However, using LDA or PLDA, which make the use of the training speaker labels to learn the transforms, brings much more favourable results to deep embeddings, especially deep CNN embeddings – the lowest obtained EER was for deep CNN embeddings with an LDA/PLDA back-end (0.39%). The comparison with the results using cosine back-end for scoring indicates the importance of employing a supervised

	EER%		
	cosine	LDA	LDA/PLDA
i-vector (FBANK)	14.76	6.93	4.55
i-vector (MFCC)	**5.71**	4.20	0.74
DNN embed.	35.71	6.06	2.25
deep CNN embed.	22.68	**1.65**	**0.39**

Table 6.6: EER (%) for i-vectors and deep embeddings evaluated in the speaker recognition task with different back-ends.

transformation into scoring.

MFCC i-vectors consistently result in a higher EER compared to CNN embeddings. This result suggests that deep CNN embeddings may be highly effective for speaker recognition task, although more experiments using a dataset with more speakers would be necessary to confirm this finding. In Section 6.4.2 we evaluate the AM embeddings for AMI, and in Section 6.4.3 – for VoxCeleb.

Acoustic condition recognition

To learn whether deep embeddings contain sufficient information to differentiate between 14 acoustic conditions (utterances with 7 different types of noise added, recorded with matched or mismatched microphone) we also perform an acoustic condition recognition task. The acoustic conditions in the training set are of the same type as the conditions in enrol and eval datasets, however the noise was added to each utterance at randomly chosen 5-15 dB SNR level for the eval and enrol utterances and 10-20 dB SNR level for the multi-condition training set. This introduces variability to the data and hence the acoustic conditions in the training set are not fully matched to the enrol and eval sets. In this experiment, we average utterance-level vectors across the acoustic conditions to obtain enrolment vectors. Table 6.7 shows the EERs for all four types of embeddings using cosine distance and speaker informed LDA transforms.

The lowest EER (10.13%) was achieved using a raw DNN embedding, showing that the representations learned by a DNN model contain the most information about the actual acoustic noise type. Deep CNN embeddings also seem to be able to differentiate between acoustic conditions well; however, i-vectors perform more poorly. When we transform the embeddings using the LDA speaker transform, all

	cosine	LDA (10)	LDA (70)
i-vectors (FBANK, 400)	25.97	42.51	31.60
i-vectors (MFCC, 400)	16.15	45.50	24.50
DNN embed. (400)	**10.13**	45.89	31.21
deep CNN embed. (400)	10.95	47.88	38.87

Table 6.7: EER (%) for i-vectors and deep embeddings evaluated in the acoustic condition recognition task. LDA is informed by both genders speaker labels.

of the representations lose their ability to differentiate between acoustic conditions to some degree. The LDA experiments were carried out with a dimensionality of 10 (optimal for LDA gender independent scoring of deep CNN embeddings for speaker recognition) and of 70 (optimal for gender independent scoring of MFCC i-vectors). As expected, reducing the dimensionality of the embeddings with speaker LDA transform results in more speaker-informative and less domain-informative embeddings.

Comparing the results using the cosine distance back-end for speaker and acoustic condition recognition within the same embedding type indicates which attributes of the speech signal are modeled by those representations. Both MFCC and FBANK i-vectors are more speaker specific than noise specific, with MFCC i-vectors being much better for both tasks compared to FBANK i-vectors. However, the results show the opposite for deep embeddings: both DNN and deep CNN embeddings result in a lower EER for acoustic condition recognition. We used this observation to improve the quality of the extracted embeddings for acoustic model adaptation (in Chapter 4) by forcing the embeddings to be more similar to i-vectors in terms of the encapsulated information (i.e. more speaker-aware and less noise-aware) with the LDA speaker transformation prior to adaptation.

The main difference between DNN and deep CNN representations is in their ability to characterize speakers, with deep CNN embeddings being superior for this task. This confirms our initial t-SNE visualisations conclusions and suggests that the deep CNN acoustic model might perform better in the ASR Aurora-4 task compared to the DNN model (9.55% WER compared to 12.55% WER) because of the deep CNN model's ability to learn more speaker-aware intermediate representations.

Noise and gender recognition

For completeness, we evaluate the embeddings in noise recognition (seven classes) and gender recognition (two classes). The results are in Table 6.8. We do not use LDA/PLDA scoring, because our primary goal is to investigate the information encoded in the utterance embeddings and look for the differences between the models, rather than optimising the performance for the downstream speech attributes recognition tasks.

	noise	gender
DNN embed. (400)	**7.32**	38.10
deep CNN embed. (400)	9.18	**24.50**

Table 6.8: EERs [%] for cosine scoring for noise and gender verification tasks.

Those results confirm that DNN internal representations are more aware of the background acoustic noise (acoustic condition and noise recognition scores), and deep CNN representations are more aware of the speaker related characteristics (speaker and gender recognition scores). We hypothesise that there might be a relation between those characteristics of the models and their final ASR performance. We next investigate layer-by-layer characteristics of the models, to better understand their behaviour.

Layer-wise representations

We extract layer-wise representations, to study the encoding properties of learned representations in various layers of the network. By looking at the discriminative power of these representations we aim to learn more about the learning process of the deep CNN and the DNN models. Besides the comparison between different models, it is also interesting to compare the layer-specific embeddings within the same model, to see how the information about different attributes is propagated in the networks. With those analyses we aim to evaluate which underlying factors of variation are relevant for speech recognition, and at which layers in the network. We are also looking to learn if the representations are disentangled for the underlying factors of variation, and what the layer-wise breakdown is.

To obtain layer-specific embeddings, we pool the frame-level activations at five CNN and six DNN layers. We provide the EER score for input and output

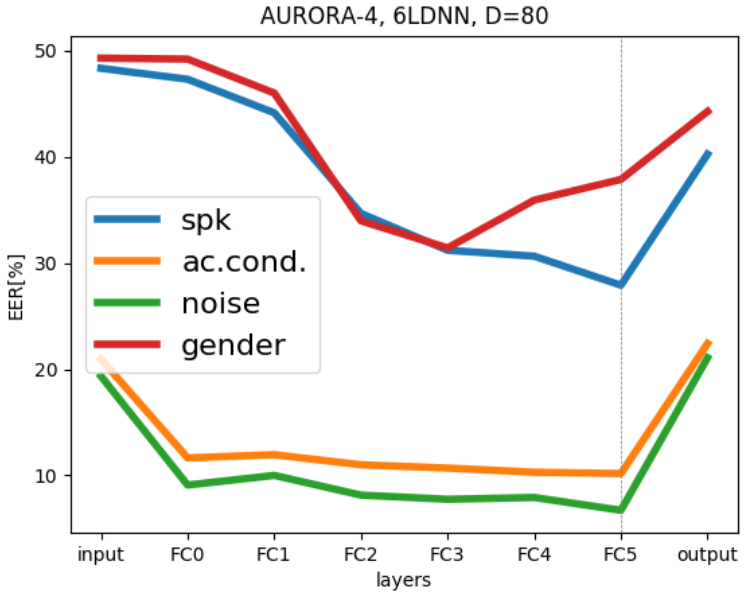


Figure 6.7: Layer-by-layer EERs [%] with cosine back-end for a 6LDNN model in speaker (spk), acoustic condition (ac. cond.), noise and gender recognition. The dimensionality of the layer-wise embeddings D=80.

representations as well, which were obtained similarly to the utterance-level embeddings – by averaging frame-level representations for each utterance. For the input representations, the frame-level representation is flattened FBANK feature map, for the output representations we used the output of the last layer, before the softmax operation. We evaluate the representations in the speaker, acoustic condition, noise type, and gender recognition. We hypothesize that input and output representations would be in general less characteristic of those attributes than the intermediate representations.

We choose to keep the dimensionality of layer-specific embeddings constant (80), however we also examined the number of components needed in the layer-specific PCA representation in order to retain 99% of its descriptive power. For the DNN model, the proportion of components required in subsequent fully-connected layers is 1%, 3%, 8%, 14%, 21%, 31%, respectively. This confirms that the representations in the upper layers are richer, thus more components are needed to represent the same amount of variability. This observation is also confirmed for the deep CNN model, with 1%, 1%, 3%, 6%, 14% of components respectively needed to retain the variability at a constant 99%.

The results for DNN layers are shown in Figure 6.7, and the representations

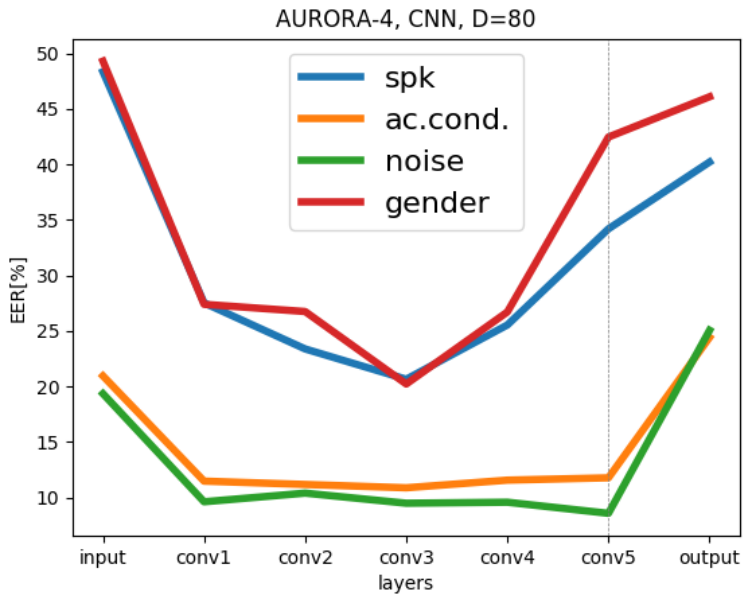


Figure 6.8: Layer-by-layer EERs [%] with cosine back-end for a 15LCNN model in speaker (spk), acoustic condition (ac. cond.), noise and gender recognition. The dimensionality of the layer-wise embeddings D=80.

for the last convolutional layer in each of the five convolutional blocks of the deep CNN (i.e. the output of layer 3, 6, 9, 12 and 15 of *deep CNN-allconv*) are in Figure 6.8. The results for the embeddings of variable number of components (but with the constant variance explained) followed the same pattern as fixed size embeddings.

The deep CNN model learns about speaker and gender characteristics more effectively than the DNN model. After only the first convolutional block (3 layers) the EER drops from 48.35% at the input to 27.53% for speaker recognition. For the DNN model, the EER after 3 layers drops to 34.68%.

The ability of the layer-specific embeddings to characterise speakers degrades after the third convolutional block (the EER starts to increase). This characteristic is what differentiates the CNN and the DNN plots. For the CNN model, the speaker curve decreases until the conv3 layer, then increases towards the output. The DNN speaker curve only decreases (if we do not consider the output representation). This finding indicates that the deep CNN network is implicitly performing speaker normalisation – the network has learned the differences stemming from different speakers (decreasing part) and removed them (increasing part). This behaviour can be optimal for SI ASR – the model initially learns

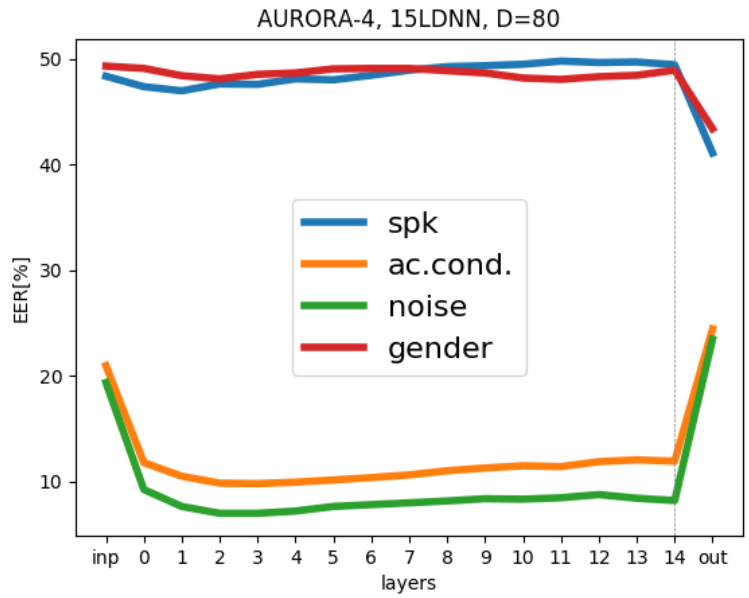


Figure 6.9: Layer-by-layer EERs [%] with cosine back-end for a 15LDNN model in speaker (spk), acoustic condition (ac. cond.), noise and gender recognition. The dimensionality of the layer-wise embeddings D=80.

about the speakers, to then solve the primary senone classification task. The curve for the DNN suggests that the parameters in the final fully-connected layers are used to learn speaker differences, when perhaps the upper layers in the network should be putting more resources into senone classification for optimal performance.

Relating layer-wise representations to whole model embeddings explored previously in this section, the majority of the highest variance components in the whole model embedding are from the last layer of the CNN model – conv5. Since the variance in this layer is not mostly due to speaker characteristics (as indicated by high speaker EER in this layer) but perhaps the phonetic information in the utterances, it can explain why the deep CNN performed better than the DNN. For the DNN model, the vast majority of the highest variance components are also in the last layer, but the representation at the last layer is not speaker-invariant – the ability to differentiate between speakers is the highest at the last layer – which might explain worse performance compared to the deep CNN model. We argue that the representations close to the output classification layer should not be speaker-specific, to enable robust and speaker-independent speech recognition.

It seems that the gender normalisation is performed by both the CNN and the

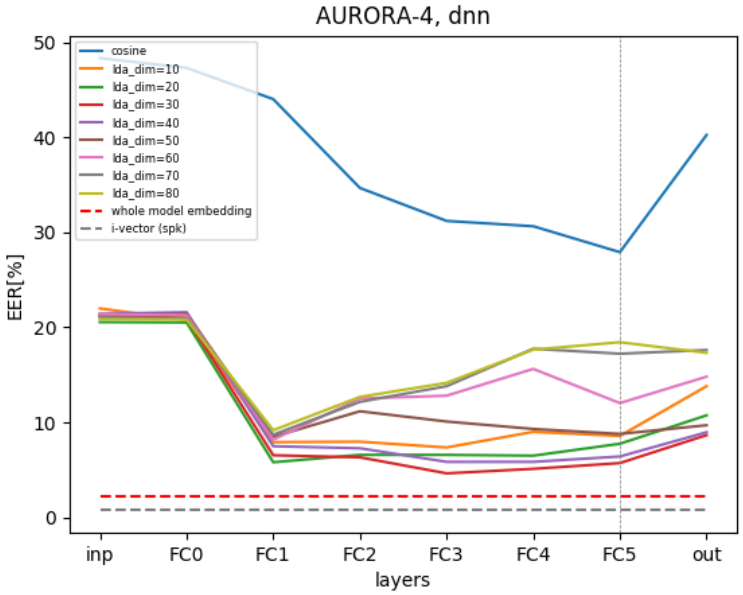


Figure 6.10: Performance in speaker recognition task measured in EER[%] with cosine and LDA/PLDA back-end for embeddings extracted from different layers of a 6-layer DNN trained on multi-condition Aurora-4 set. WER for this model is 12.55% for all test sets.

DNN models. This result might indicate that gender normalisation is an easier task than speaker normalisation. However, the deep CNN model is more effective than the DNN model in gender normalisation (lower gender curve minimum value and higher value at the last layer). This might also contribute to its superior ASR performance.

Noise and acoustic condition normalisation appears to be similar in both models. The EERs are lower than for speaker and gender recognition. This might indicate that acoustic condition and noise normalisation is an easier task. On the other hand, the yellow and green curves are non-increasing across hidden representations. Therefore, the output representations can be noise-dependent.

For comparison, we also show the plots for a fifteen layer DNN model. Its WER performance was 21.91% – much worse than the six layer DNN and fifteen layer CNN analysed previously in this section. Figure 6.9 shows layer-by-layer EERs for this model. We can see that the main difference between this and the previous models is in speaker and gender curves. We hypothesise that the inability of the fifteen layer DNN model to normalise out speaker-related features is one of the factors contributing to overfitting and its sub-optimal performance.

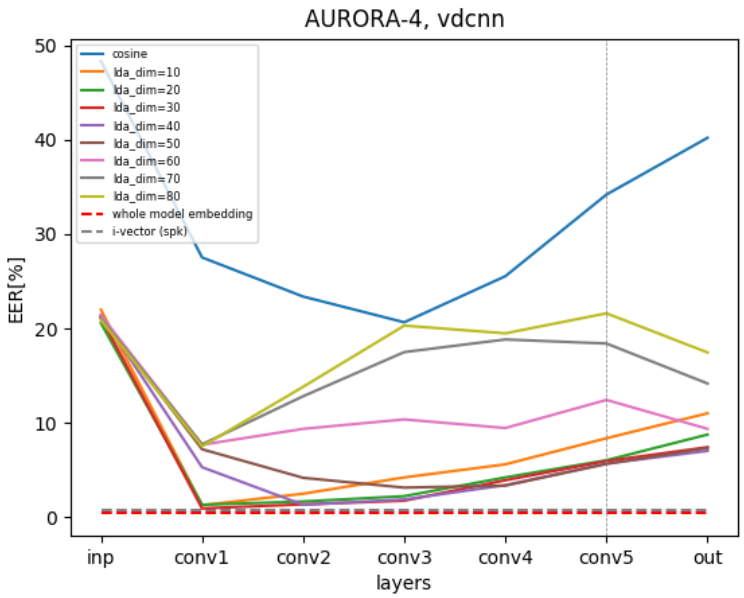


Figure 6.11: Performance in speaker recognition task measured in EER[%] with cosine and LDA/PLDA back-end for embeddings extracted from different layers of a CNN trained on multi-condition Aurora-4 set. WER for this model is 9.55% for all test sets.

The flat speaker and gender curves indicate the inability of the model to normalise out the speaker identity from input representations. The network did not learn to disentangle the underlying factors of variation (speaker and gender); instead, it could solve the task by memorising the representations for particular speakers, rather than by learning universal senone representations.

We next concentrate on speaker discriminability and we compare cosine scores to LDA/PLDA scores for layer-wise DNN and CNN representations (Figures 6.10-6.12). We also show the performance of the whole model embeddings and the i-vectors for speaker verification on the plots.

As expected for the DNN (Figure 6.10), increasing the dimensionality of the LDA for LDA/PLDA scoring causes the speaker curves to exhibit the behaviour observed previously only for the CNN representations – the speaker curve is increasing towards the output. Another interesting observation is that none of the layer-wise embeddings outperformed the whole model embedding for speaker verification. This holds for both the CNN embeddings (Figure 6.11) and 15-layer DNN embeddings (Figure 6.12). This indicates that combining the activations from all layers in the whole model might be beneficial for speaker verification. We evaluate this hypothesis for AMI and VoxCeleb in the next sections.

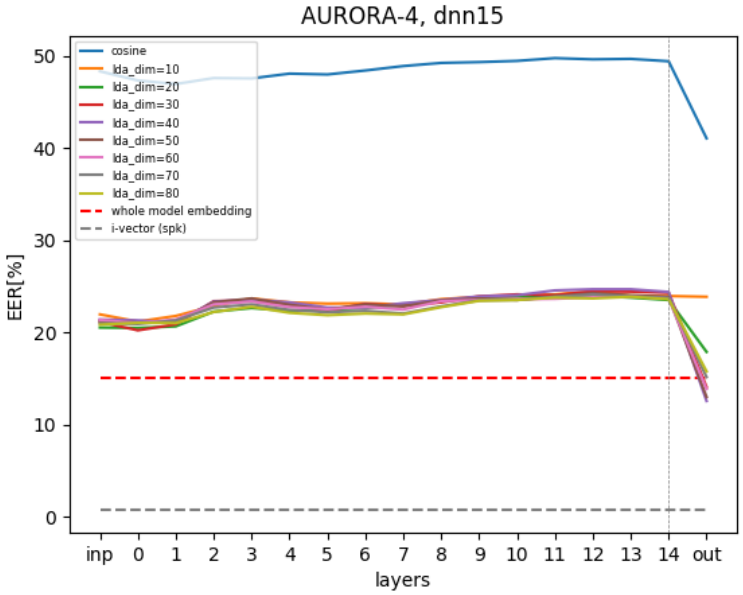


Figure 6.12: Performance in speaker recognition task measured in EER[%] with cosine and LDA/PLDA back-end for embeddings extracted from different layers of a 15-layer DNN trained on multi-condition Aurora-4 set. WER for this model is 21.91% for all test sets.

The patterns for the CNN (Figure 6.11) and the 15-layer DNN (Figure 6.12) stay similar, regardless of the scoring method. However, further investigation would be necessary to determine if the described models behaviour holds for the models with different training data (clean condition), with different instances of the same architectures (different initial conditions), with different number of parameters, with alternative architectures (e.g. TDNNs), and for different datasets. We explore the generalisation to other datasets in the next sections.

6.4.2 AMI IHM

The main differences between the models were previously observed for the speaker recognition curves, thus we evaluate the embeddings for this task for another dataset – AMI. We use AMI dataset because it is larger and more difficult ASR task; it also has more speakers than Aurora-4. We extend the comparison of the embeddings to x-vectors.

We create our own split of data for enrolment (enrol) and evaluation (eval). We first merge the original dev and eval sets (Kaldi ASR split) to maximize the

number of speakers for evaluation (135), and then split the set into two parts, such that utterances from every speaker are found in both enrol and eval sets. We use the enrol set to obtain speaker-level representations, and we evaluate the utterance-level representations from the eval set against them, using the same scoring methods as for Aurora-4, with an addition of just PLDA scoring (no dimensionality reduction with LDA prior to PLDA scoring). We create the trials with non-target proportion 50% and with the non-matching speaker for the non-target part of trials chosen at random. We follow the same steps for the DNN and CNN embeddings extraction as for Aurora-4.

Speaker recognition

The results for speaker recognition are presented in Table 6.9. For AMI, 400 principal components were not enough to explain 99.0% of variance – 752 components would be necessary for CNN and 739 for DNN embeddings. Therefore, we use 800 components for PCA computation. Deep CNN embeddings extracted with a 800D PCA transformation are the most accurate speaker representations. Adding more dimensions is also beneficial for the DNN embeddings.

Comparing i-vectors and x-vectors, we find that i-vectors are superior with all back-ends, even when x-vector are extracted from a pre-trained SRE16 model[9], which is trained on substantially larger dataset (augmented Switchboard, Mixer 6, and NIST SREs)[10]. The best approach for x-vectors is to use the pre-trained model for the x-vector extraction, and use matched AMI data for scoring (LDA and PLDA training). With this the EER drops down to 14.51% for PLDA scoring. This result is still high, considering superior x-vector performance compared to i-vectors reported in the literature. Perhaps an optimal performance for x-vectors is dependent on matched training and test conditions, as well as a large and diverse training corpus.

We investigate the EERs for different lengths of the enrol and eval utterances (training set for PLDA training remains the same). The results are presented in Figure 6.13. We set the minimum threshold for the utterances from 0.5 to 5 seconds with the step of 0.5 seconds and we filter the utterances based on this threshold. This means, that e.g. the data points at `length > 2 sec` represents

[9] https://kaldi-asr.org/models/m3

[10] Since NIST SRE data has a sampling rate of 8kHz, we down-sample AMI data to 8kHz as well before extracting x-vectors

	EER%			
	cosine	PLDA	LDA	LDA/ PLDA
400D i-vector	**10.05**	10.93	10.98	11.88
512D x-vector (AMI training+AMI back-end)	31.47	44.95	43.27	27.85
512D x-vector (SRE training+SRE back-end)	-	-	43.47	33.83
512D x-vector (SRE training+AMI back-end)	37.36	13.42	14.65	14.51
400D 6L DNN embed.	25.94	6.92	10.16	7.18
800D 6L DNN embed.	25.90	5.97	9.24	6.60
400D CNN embed.	20.65	5.10	7.09	5.55
800D CNN embed.	20.62	**5.06**	**6.70**	**5.34**

Table 6.9: EER (%) for i-vectors and deep embeddings evaluated in the speaker recognition task with different back-ends. The dimensionality of speaker embeddings is 400 or 800, and 400 for i-vectors.

testing conditions (enrol and eval utterances) of more than two second. The trials are created for each length threshold, with a constant 50% non-target proportion, and trial pairs chosen at random. The speaker enrolment embeddings are also recomputed separately for each length threshold.

The plot reveals different patterns for different embeddings. For the testing conditions with short utterances, CNN embeddings are superior to i-vectors and x-vectors. For longer recordings, x-vectors start to outperform both i-vectors and deep CNN embeddings in the speaker recognition task. CNN embeddings perform similarly across different lengths of utterances. We hypothesise that this is due to small convolutional kernels in the CNN embeddings extractor architecture, which exploit local dependencies in the acoustic time-frequency representations. Combining local representation learning with segment-level mean pooling[11] and speaker-level PLDA back-end makes the CNN embeddings effective in the speaker recognition task across different lengths of utterances.

We also show the EERs for different length ranges of the enrol and eval utterances, rather that with the minimum length threshold. We do this to analyse the length bins independently and provide more insight on the influence of the utterance length on the embedding performance. As expected, the data points

[11]Refer to Fig. 4.5 in Chapter 4 for the CNN embeddings extraction framework.

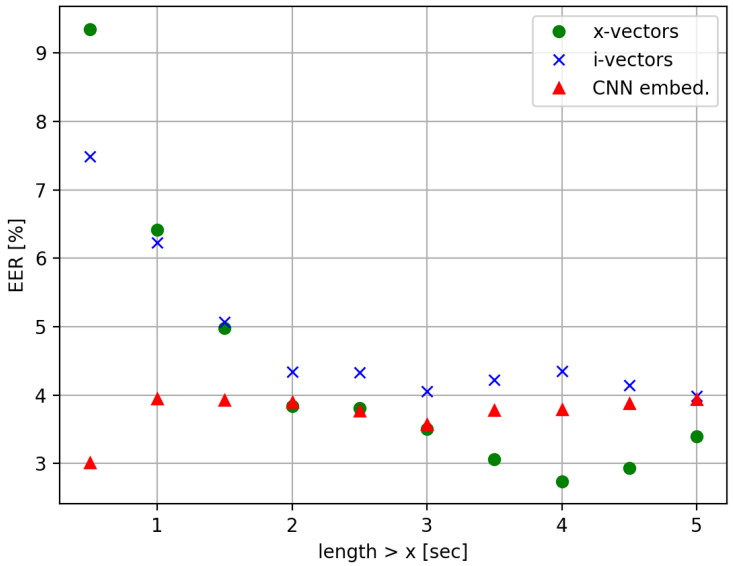


Figure 6.13: EERs (%) for x-vectors, i-vectors and CNN embeddings with PLDA scoring for speaker recognition. enrol and eval recordings are filtered by the minimum length.

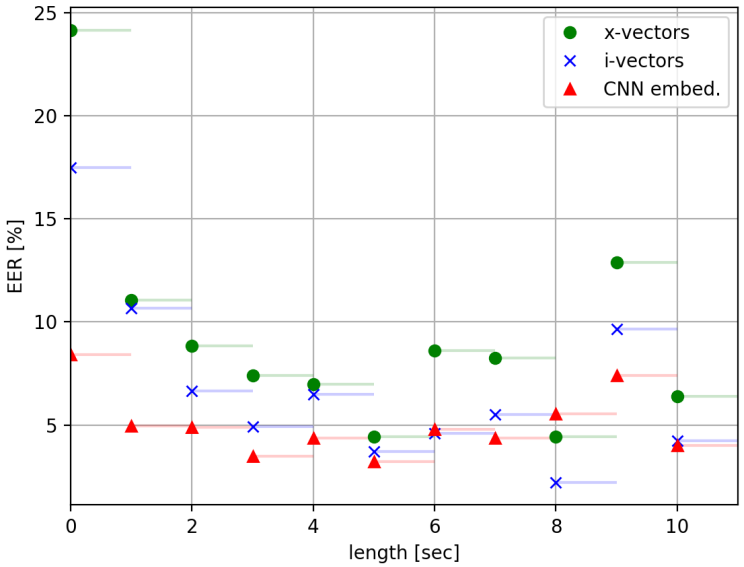


Figure 6.14: EERs (%) for x-vectors, i-vectors and CNN embeddings with PLDA scoring for speaker recognition. enrol and eval recordings are filtered according to the length range.

are more noisy in this case (Figure 6.14) than in the previous figure. We still observe superior deep CNN embedding performance, especially for testing con-

	EER%			
	cosine	PLDA	LDA	LDA/PLDA
i-vector (MFCC)	**14.99**	13.13	12.83	13.54
400D 6L DNN embed.	35.95	6.96	9.36	6.89
800D 6L DNN embed.	35.91	6.27	8.43	6.18
400D CNN embed.	31.37	3.76	4.72	3.55
800D CNN embed.	31.32	**3.46**	**4.36**	**3.19**

Table 6.10: EER (%) for i-vectors and deep embeddings evaluated in the speaker subset verification task with different back-ends, where speakers are split such that there is no more than thirty seconds of data per speaker.

ditions with short utterances, however, using only very short utterances is not better than using only longer utterances. The minimum EER for deep CNN embeddings is in the range 5-6 seconds, and for x-vectors and i-vectors – in the range 8-9 seconds. Overall, deep CNN embeddings are less venerable to length changes, making them potentially more robust speaker embeddings.

We believe that the extraction of deep CNN utterance-level embeddings from a fully-convolutional architecture, with kernels spanning over time and frequency, enables the capture not only of speaker characteristics, but also of more local time and frequency acoustic patterns, which was first observed with t-SNE visualisations. This hypothesis is also supported by the results in Table 6.10, where instead of speaker verification, we perform speaker subset verification. Speaker subsets are created by splitting within existing speakers into several groups, with a thirty seconds of data per subset restriction. Contiguous utterances are assigned to the subsets. As in the previous experiment, the opposite trend can be observed for deep CNN embeddings than for the i-vectors – EERs are lower for CNN embeddings and higher for i-vectors for speaker subsets, compared to genuine speakers. Therefore, deep CNN embeddings are not only good at capturing speaker characteristics, but can also represent more local characteristics of the utterances, potentially corresponding to different acoustic conditions, channel characteristics, or phonetic content of the utterances. We next analyse how those characteristics correspond to the performance of the embeddings in DNN-SAT.

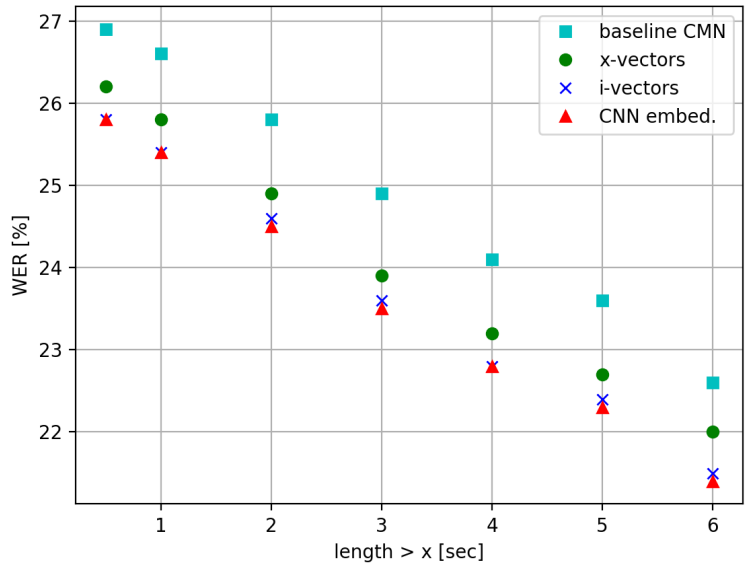


Figure 6.15: WERs (%) for a baseline CMN model and models with x-vectors, i-vectors and CNN embeddings used in SAT-DNN filtered with minimum length threshold.

Relation to ASR performance

In Chapter 4 we explored the auxiliary embeddings for acoustic model adaptation. Here, we present the WERs for the models trained with different embeddings (for a SAT-DNN model from Table 4.9) for test utterances filtered with minimum length threshold (Figure 6.15), as well as according to the length range (Figure 6.16).

Those plots can be compared side-by-side with the corresponding EER plots to be able to relate the performance of the embeddings in the speaker verification task and in the acoustic model adaptation task. It is interesting to see that even though deep CNN embeddings are better speaker representations for short utterances (Figure 6.13), they do not contribute to better performance in SAT-DNN for short utterances (Figure 6.15), compared to other representations. There is a big gap between deep CNN embeddings and i-vectors for speaker verification for short utterances, yet those two representations perform similarly for adaptation.

Regardless of the utterance length, i-vectors and deep CNN embeddings perform similarly for adaptation. For x-vectors, even though they outperform the other embeddings in a speaker recognition task for longer utterances (more than 3.5 sec), they do not outperform the other embeddings when used for SAT-DNN.

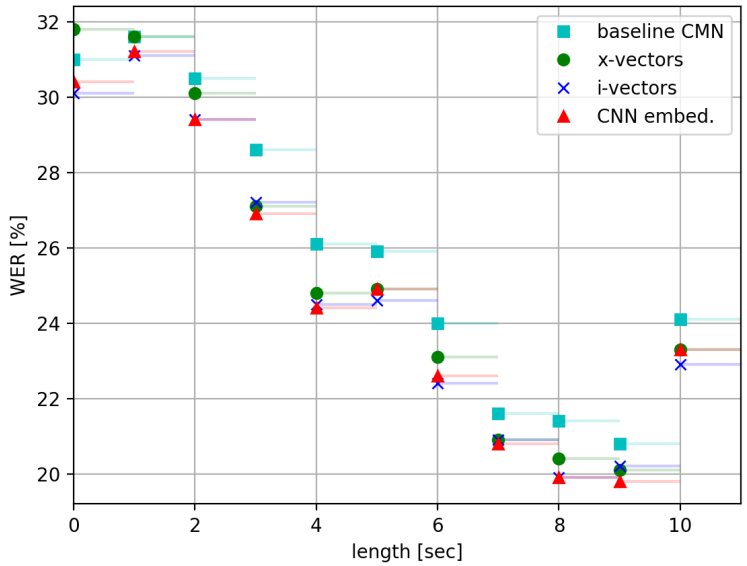


Figure 6.16: WERs (%) for a baseline CMN model and models with x-vectors, i-vectors and CNN embeddings used in SAT-DNN filtered with length range.

The analyses confirm that better speaker representations do not guarantee more effective normalisation in SAT-DNN. Perhaps i-vectors and deep CNN embeddings can better capture other information alongside speaker identity.

The analysis in Figure 6.16 reveals that when decoding only short utterances (less than 2 sec), the baseline without any adaptation is superior to the model using x-vectors as auxiliary information. This shows that the quality of the embeddings used for adaptation is the most important for short utterances. Perhaps this observation could be used to switch to decoding without an auxiliary embedding for short utterances. However, what the *quality* of the auxiliary embeddings corresponds to remains an open question, since better speaker discriminability can only partially explain differences in WER performance.

We next look at layer-by-layer activations and test them for speaker verification. We are interested to see if the same within and between models patterns can be observed for AMI as for Aurora-4.

Layer-wise representations

The extraction framework of the layer-wise representations is the same for AMI as for the Aurora-4 task.

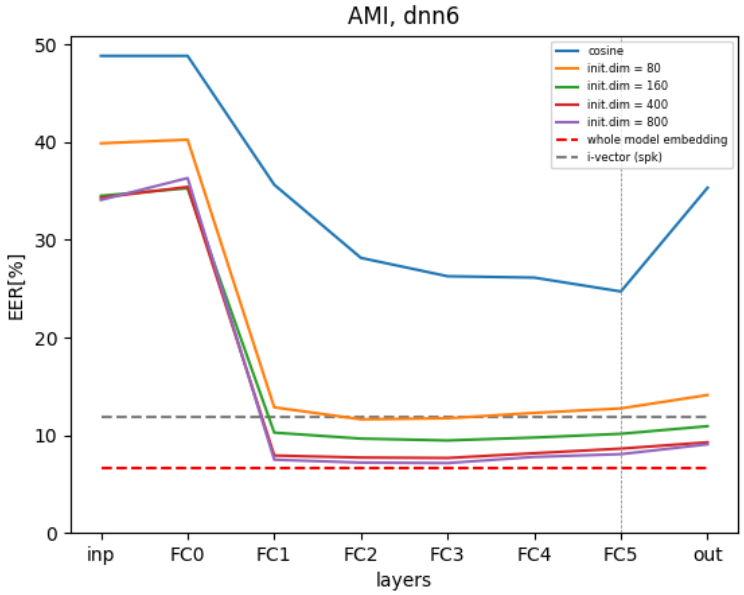


Figure 6.17: Performance in speaker recognition task measured in EER[%] with LDA/PLDA back-end for embeddings extracted from different layers of a 6-layer DNN, with different initial dimensions. WER for this model is 27.9% (IHM).

The speaker verification curves for the DNN (Figure 6.17) and the CNN models (Figure 6.18) show similar patterns to Aurora-4 plots. The DNN curve does not exhibits the U-shape for the hidden representations, as does the CNN curve[12]. We vary the initial dimension[13], i.e. the number of components of the PCA transform, and find that it does not influence the shape of the curves a lot. LDA dimensions were optimised separately for each layer and each initial dimension. The layer-wise PCA dimensionality had to be increased to retain 99.0% of variance, indicating more varied hidden representations for AMI than for Aurora-4.

The figures also show whole model embeddings with 800 dimensions, and per speaker i-vectors, for comparison. Interestingly, for the DNN, the lowest EER per layer (FC2) is 7.15%, while for the whole model embedding it is 6.60%. Similarly for the CNN, the lowest EER per layer (conv3) is 5.43% and for the whole model embedding it is 5.34%. This result is in line with the previous findings, showing that merging the representations across the layers could provide a better speaker embedding. We evaluate the CNN embeddings for a larger speaker verification

[12]The architecture for this model is the same as for the CNN baseline in Chapter 5 (Table 5.3) but here we train on IHM instead of MDM AMI scenario.

[13]The maximum initial dimension for the input is 440, thus for the input at init.dim = 800 the actual number of dimensions is 440.

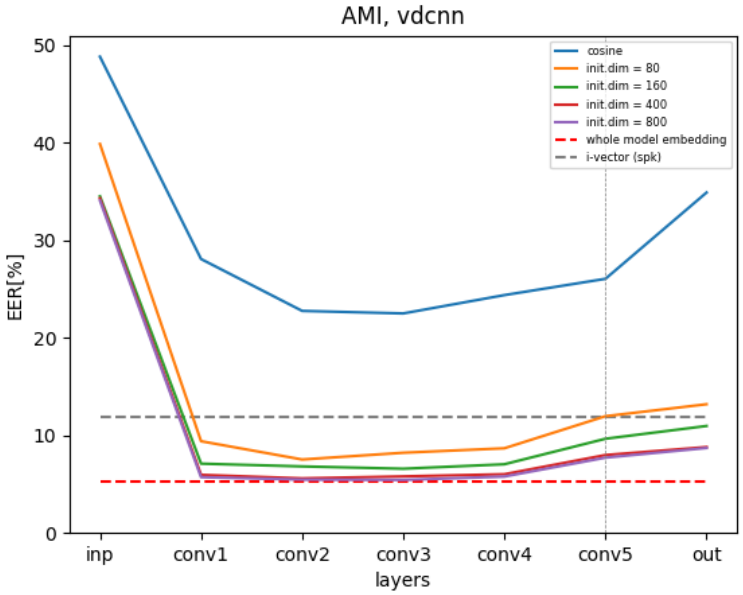


Figure 6.18: Performance in speaker recognition task measured in EER[%] with LDA/PLDA back-end for embeddings extracted from different layers of a CNN, with different initial dimensions. WER for this model is 26.1% (IHM).

task (VoxCeleb) in the next section.

6.4.3 VoxCeleb1

We have observed very good speaker verification performance for our deep CNN embeddings, therefore we test them for a common speaker verification dataset – VoxCeleb1 (described in Chapter 3). In this section, the goal is to obtain the most discriminative speaker embeddings. In our experiments, we use the deep CNN acoustic model trained on AMI IHM data (the same model as in Section 6.4.2) to extract the utterance embeddings for VoxCeleb1 dev and test sets (verification split). We use the official trial pairs for scoring. The dev set is used for LDA and PLDA training, while the PCA is trained on AMI IHM dataset. We extract the PCA embeddings for 512 components (to match the dimensionality of the x-vectors), for 800 components (as this number was necessary to explain 99.0% of variance for AMI), and 1200 components (to test the 99.0% of variability assumption).

The results are presented in Figure 6.19. We extract two versions of the embeddings – with and without silent frames. Silence removal is a common

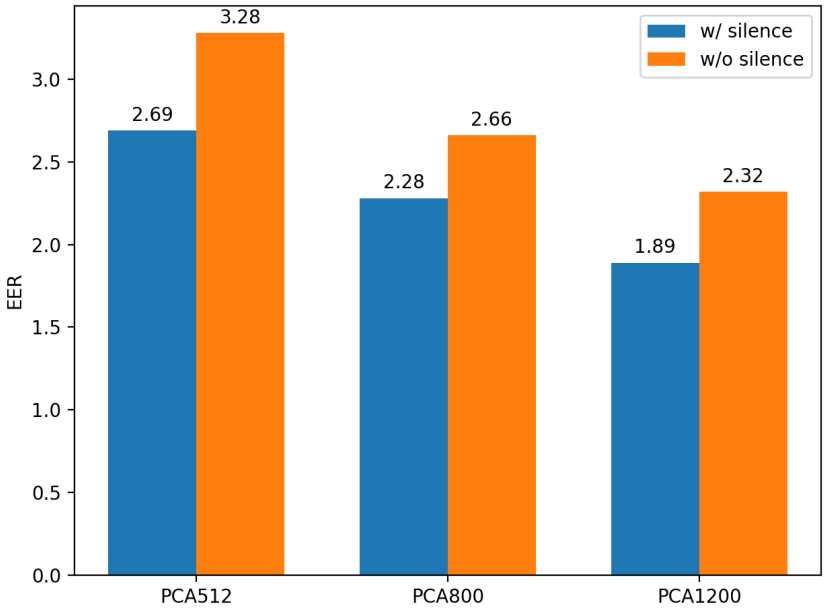


Figure 6.19: EERs for VoxCeleb1 test set for deep CNN embeddings with different number of PCA components. "w/o silence" results were obtained by removing silent frames before mean normalisation per speaker.

pre-processing step for speaker embedding extraction, when the best speaker discriminability is the main objective. We hypothesise that for our embeddings, silence removal might not be beneficial, because the embedding extractor, which is an acoustic model, can explicitly model silence with dedicated HMMs. We empirically confirm that removing silent frames before deep CNN embeddings extraction resulted in higher EERs. The dimensionality of the LDA transform was optimised separately for each embedding type – it was 120 for the best embedding in Figure 6.19. The other interesting observation is that increasing the number of PCA components to 1200 is beneficial, indicating that 99.0% of variance threshold might not be optimal for the embeddings for speaker verification.

The obtained EERs are low, compared the results in the literature. For example, Nagrani et al. [2017] report 7.8%, Snyder et al. [2018] – 3.10%, and recently Nagrani et al. [2019] – 2.87% for the same test set. However, those results can not be compared directly. The advantage of the mentioned works is in matched and larger training sets. They either train on VoxCeleb1 (352h) or VoxCeleb2 (2,442h), whereas we train the embedding extractor on AMI IHM (78h).

The advantage of our framework is in per speaker mean normalisation of the input features (see Figure 6.4). For speech recognition, per speaker mean

PCA training	EER%
AMI IHM train	7.20
VoxCeleb1 dev	6.87

Table 6.11: EERs for VoxCeleb1 test set for deep CNN embeddings for PCA with 1200 components and per utterance mean normalisation. The PCA transform is trained on AMI IHM training set or VoxCeleb1 dev set.

normalisation is performed to enhance speaker independence of the features, to learn robust senone representations. For speaker recognition, sliding window normalisation is typically employed [Alam et al., 2011] (at train and test time), to compensate for the effects of environmental mismatch. In our embedding extraction framework we used per speaker mean normalisation. This is an advantage because we use all per speaker data, even at test time. It is not a viable setup for speaker verification when we can not use per speaker statistics before recognising the speaker at test time. On the other hand, it is interesting that with per speaker normalisation we were able to extract very effective speaker embeddings. We hypothesise that with speaker-wise input feature normalisation we removed the most obvious differences between speakers, which enabled the extractor to learn more subtle differences between them. To evaluate the embeddings in more realistic speaker verification setup, we replace per speaker normalisation with per utterance normalisation; however, we do not change the acoustic model – it is trained with per speaker mean normalisation. We again optimise the LDA dimensionality separately. We also explore the effect of matched data used for PCA training. Table 6.11 shows EERs for the embeddings extracted with per utterance normalisation over the input features and with PCA trained on AMI training set (as for the speaker normalised results), and with PCA trained on VoxCeleb1 dev set.

Mean normalisation per utterance degrades the performance but it is a straightforward way to make the embedding extraction framework more applicable for the speaker verification task. Using a matched dataset for PCA training is beneficial, with the EER of 6.87%. This number is not as good as state-of-the-art results for VoxCeleb, but given mismatched and relatively small NN training set, we believe our deep CNN embeddings are competitive with other state-of-the-art speaker embeddings.

To further improve the results, re-training the embedding extractor with per utterance or sliding window mean normalisation could be worth trying, to ensure matched normalisation approach at all embedding extraction stages. Also, using a training set (for NN training) with more speaker diversity could be beneficial. We leave this as future work.

6.4.4 Conclusions

In this section, we described the analyses of the representations learned by well-performing acoustic models. We used verification techniques to decode the information about different speech attributes from internal representations, to learn about the information implicitly acquired by the models. We extracted whole model embeddings, to learn about model-specific encoding properties, as well as layer-wise embeddings, to show where speech attributes specific information in networks is formed. We compared the embeddings between a CNN and a DNN acoustic model, as well as between different layers of the same network. We performed the analyses for Aurora-4, AMI, and VoxCeleb datasets.

We found that the CNN model implicitly learns features useful for speaker recognition. The speaker discriminability curves are different for the DNN and the CNN models, but consistent across Aurora-4 and AMI. It seems that the CNN performs the speaker normalisation more efficiently than the DNN. We also found for both Aurora-4 and AMI that merging the activations across layers contributed to better speaker embeddings, compared to activations from any single layer. Evaluation for VoxCeleb confirmed that the proposed speaker embedding extraction framework is competitive with other deep speaker embeddings in the literature.

The analyses in this section confirmed that the CNN model learns local acoustic correlations. This was expected, since the CNN model used in this section has a VGG-like architecture, with small 2D kernels and local receptive fields. We hypothesise that the local acoustic correlation learning can be one of the reasons for superior CNN performance for both ASR and ASV.

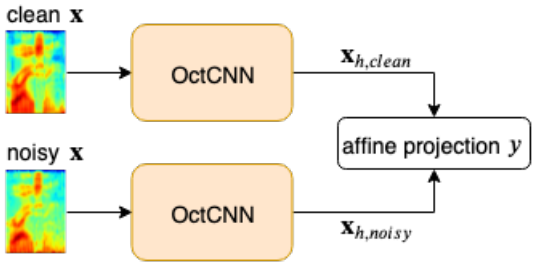


Figure 6.20: Proposed method to measure the robustness of learned representations.

6.5 Comparison of learned representations

In this section, we propose a method to compare learned representations with the objective to measure the robustness of the layers and whole models. The existence of robust layers in the model could imply better generalisation. With this analysis we aim to provide more insight into the models, to better understand learned representations. We perform the analyses for OctCNN models, described in Chapter 5, trained on Aurora-4 multi-condition training set[14]. We compare the representations between models (OctCNN and vanilla CNN), as well as within models, for low and high frequency representations.

Instead of directly comparing two representations, we train a projection matrix with the objective to map one representation into the other, and we treat the cost of this mapping as a similarity measure (lower cost means higher similarity). Therefore, this approach allows for comparison of the representations of different dimensions, but it requires training an auxiliary network for each comparison.

We compare the projection of clean samples with time-aligned noisy samples. The overall process is shown in Figure 6.20. The similarity between the projections is measured using the mean squared error (MSE) loss of an affine projection y of N clean to noisy samples (Eq. 6.5). Instead of a direct comparison between clean and noisy encodings, an affine projection is used to take into account permutations of hidden representations, i.e. to ensure invariance of the metric to affine transformations of the encodings. This allows for a robust comparison between clean and noisy samples, as well as between models and layers (or group

[14]We analyse the best performing OctCNN for Aurora-4, i.e. the model with batch normalisation after ReLU, and with octave convolutions applied with [0.125, 0.875] ratio to layers L2-L15.

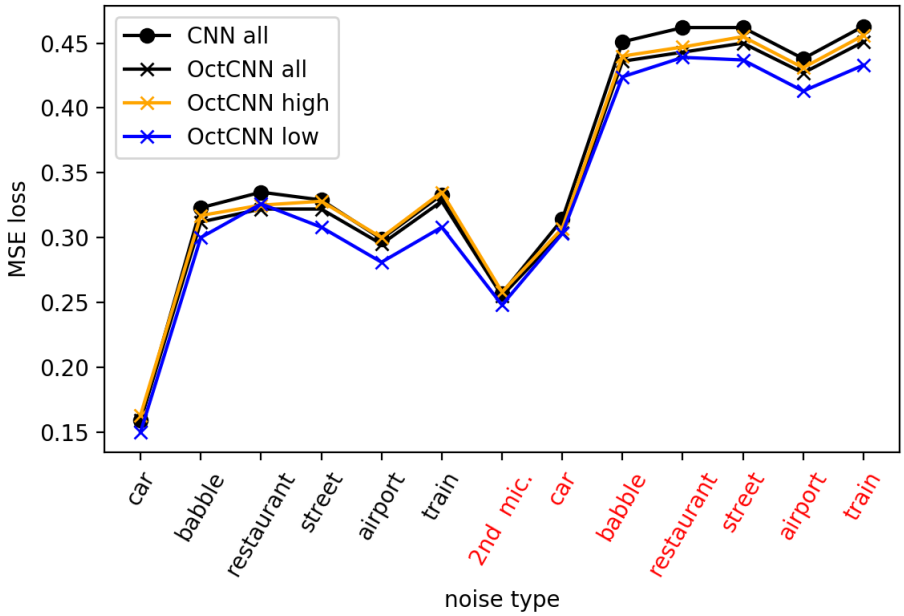


Figure 6.21: MSE affine transformation loss to measure the similarity of "clean" and "noisy" encodings ($\boldsymbol{x}_{h,clean}$ and $\boldsymbol{x}_{h,noisy}$). "all" corresponds to the output of the last convolutional layer (Conv15), "high" and "low" correspond to its $Y^{1\to1}$ and $Y^{2\to1}$ branch, respectively.

of layers).

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{ND} \sum_{i=1}^{N} \left\| y(\boldsymbol{x}^{(i)}_{h,clean}, \theta) - \boldsymbol{x}^{(i)}_{h,noisy} \right\|^2 \tag{6.5}$$

The projection y is a single fully-connected layer with 1024 nodes and without any activation function. We use the Aurora-4 test sets and compare clean encodings $\boldsymbol{x}_{h,clean}$ with noisy encodings $\boldsymbol{x}_{h,noise}$, obtained as the activations from the last convolutional layer with a forward pass through a trained model. Both hidden representations are extracted for CNN and octave CNN (OctCNN) models in order to compare representations between the models. The dimensionality of the flattened activation vector for both model types at the last convolutional layer D is 1024. Also, for intra-model comparison, we evaluate the loss with the encodings from high and low-resolution groups (paths $Y^{1\to1}$ and $Y^{2\to1}$). This analysis aims to evaluate if the low-resolution groups for noisy samples are indeed more similar to the clean ones than the high-resolution encodings, suggesting more robust representations. We optimise the parameters of y with back-propagation using a fixed number of three epochs and we report the validation loss for Aurora-4 test sets. We treat the validation loss as a distance measure, with lower loss indicating

smaller distance (and bigger similarity) between clean and noisy samples.

The results are shown in Figure 6.21. The MSE loss for the activations at the output of Conv15 (“all") is similar for CNN and OctCNN models (black curves) for test sets B (black noise type labels) and C (red “2nd mic. label"), but lower for test set D (other red noise type labels) for OctCNN (apart from car noise). This result suggests that the OctCNN representations are more robust, because the clean and noisy inputs result in more similar hidden representations. This indicates that the OctCNN representations are more robust for test set D.

As expected, within-model comparison of the loss (“OctCNN high" and “OctCNN low" curves) show that the representations at low resolution are more similar to the clean encodings from test set A than the ones at high resolution. We believe that this effect improves the robustness of latent representations and results in a decreased WER.

6.6 Conclusions

In this chapter, we strived to better understand the representations learned by selected well-trained acoustic models. First, we empirically analysed the contribution from different components in a deep CNN acoustic model. We found that for multi-condition Aurora-4 dataset, pooling layers can be replaced with convolutional layers with increased stride, with performance gains. Removing the fully-connected layers was also beneficial. We confirmed those findings on a larger MGB-3 dataset, however, the gains were smaller for this task. For both tasks the improvements were significant at the level of $p = 0.001$. Those results were in line with previous works on image classification [Szegedy et al., 2015; Springenberg et al., 2015; He et al., 2016]. We concluded that a fully-convolutional acoustic model can be optimal for speech recognition.

We next turned to visualisation techniques, in an attempt to provide an insight into the model’s decision space. With saliency maps we performed local, example-based analysis. We have seen that the evaluated model (fully-convolutional CNN) learned to encode the location of the input features of the highest magnitude. This observation indicates that the CNN model with full weight sharing is capable of learning non-local relationships, i.e. relationships between frequency bands. With t-SNE visualisations we explored the global concepts learned by the model, by arranging the utterance summaries (embeddings) extracted from a DNN or a

CNN acoustic model in a two-dimensional space. We observed that the models learned the features corresponding to speakers and noise types, even though they were not trained explicitly to learn those concepts. The CNN representations seemed to encode more local acoustic characteristics than the DNN.

With diagnostic verification, we further explored the encoding properties of hidden representations, by analysing the utterance embeddings. We compared them within models (across different layers, with layer-wise embeddings) and between models (with whole model embeddings). We used standard verification scoring approaches, and tested the embeddings for Aurora-4, AMI IHM, and VoxCeleb1 for different speech attributes verification tasks. The layer-wise analyses revealed different speaker-specific curves for CNN and DNN models, for both Aurora-4 and AMI. This behaviour might be one of the reasons for superior performance of the CNN for ASR. We also explored the whole model embeddings, and found that they perform well for speaker verification. We confirmed this finding for VoxCeleb1. Finally, we related the performance of the embeddings for speaker verification and for acoustic model adaptive training. We found that better speaker discriminability of the embeddings did not guarantee their better performance in adaptation. However, our proposed deep CNN embeddings gave state-of-the-art results for both speaker verification and acoustic model adaptation tasks.

In the last section of the chapter on better understanding of hidden representations we analysed the representations of the previously proposed multi-scale CNN model (and compared them with a vanilla CNN). We compared the embeddings of clean and noisy Aurora-4 samples. By measuring the distance between the projection of a clean embedding and a noisy one, we determined the robustness of the model. We found that the multi-scale OctCNN model learned more robust representations than a vanilla CNN model, and that the low-resolution group of layers of the OctCNN learned more robust representations than the high-resolution group of layers. Those results were confirmed by WERs in the previous chapter.

CHAPTER 7

Conclusions

This thesis addressed deep representation learning for speech recognition in the context of one of the main challenges of ASR from the linguistic perspective – robustness to high variability of speech. High speech variability can be caused by various underlying factors. We focused on the factors related to the speaker, the environment and the channel, and evaluated if they can be informative for ASR. To learn more robust, i.e. more disentangled representations of speech, we explored two research directions: embedding-based acoustic model adaptation (Chapter 4), and multi-scale representation learning (Chapter 5). We focused on feed-forward models, mostly CNNs, in a hybrid ASR framework. The proposed approaches enabled for fast and unsupervised adaptation to testing conditions, without a disjoint adaptation dataset. We also analysed learned representations to improve their interpretability (Chapter 6). In this chapter, we summarise the findings, discuss limitations and propose the directions for future work.

7.1 Summary of findings

Embeddings for attribute-aware and adaptive training

From our experiments with an auxiliary embedding, we conclude that embedding-based acoustic model adaptation is an effective approach to learn more robust

representations, and that it can be optimised. The aspects that had an influence on the performance were: the type of the embedding, the transformation of the embedding, and the level of incorporation of the embedding.

First, we evaluated how does the type of the auxiliary embedding influence the ability to learn robust representations. We looked for an optimal embedding, to provide the acoustic model with a useful feature vector, in attribute-aware and adaptive training strategies. The proposed AM utterance embeddings extracted with the speaker-informed LDA transform were optimal in a SAT-DNN framework, indicating the relevance of modelling speaker variability for robust ASR. Moreover, we found our deep CNN embeddings to be better than i-vectors for speaker and acoustic condition verification. Yet, this was not sufficient to outperform them in AM adaptation. We also found that other neural speaker embedding, i.e. x-vectors, fell short for the acoustic model adaptation task. Those results indicate that a useful auxiliary embedding should be characteristic of the speakers, but an optimal speaker embedding might not be optimal for acoustic model adaptation.

Next, we examined different transformations of the embeddings, in the search for an optimal embedding incorporation approach, to help to reduce the sensitivity to uninformative variance more effectively. We found the transformation of the embedding to have a bigger impact on WER than the type of the embedding. Using a control layer to transform the i-vector before incorporation to the acoustic model was the most advantageous in our experiments. On the other hand, using more complex auxiliary networks acting on the auxiliary embeddings did not bring further improvements. We also found embedding-based normalisation of the input feature space to be more effective than the normalisation of hidden representations.

Multi-scale representations for robust and efficient modelling

Our second line of research concerned multi-scale representation learning, with the same objective – improved robustness of the acoustic model. With this approach, we aimed to implicitly incorporate the knowledge about the underlying factors of variation, without imposing the source of the factors, by learning at multiple scales.

The results showed that multi-scale convolutional representation learning can be effective for increased robustness against different types of noises. We also

showed that tuning the ratio of the representations at different scales is important for optimal performance. We confirmed the increased invariance of learned representations to different noise types and different channels with the affine transformation loss of the mapping of clean to noisy samples, to compare their similarity. We did not, however, achieve an improved robustness against reverberation for AMI corpus. We also did not observe higher performance gains of multi-scale representations with alternative input representations, and we did not observe significant gains with data augmentation.

Although our primary goal was improved robustness measured by WER, the proposed MultiOctCNN acoustic model was also more computationally and memory efficient.

Analysing learned representations

We found that a fully convolutional model with small (3×3) 2D kernels, using increased stride to confer the invariance to uninformative local input translations, was optimal in terms of WER in our experiments. This simplified model served as a baseline for multi-scale representations, and as the embedding extractor for attribute-aware and adaptive training.

From t-SNE visualisations and diagnostic verification of whole model embeddings, we learned that the CNN model was able to learn local acoustic patterns more effectively than a DNN. Local spectro-temporal patterns can be linked to different factors of variation, such as acoustic conditions and various speakers. With LDA and PLDA transformations, we improved speaker differentability of the hidden representations, alongside their usefulness for acoustic model adaptation as the auxiliary embeddings. Hence, we conclude that speaker-related underlying factors of variation are informative for speech recognition in multi-condition training strategy. On the other hand, we did not link better acoustic condition discriminability with improved AM adaptation performance.

Analysing learned representations for the underlying factors of variation related to the speaker and the environment revealed similar patterns of speaker awareness for Aurora-4 and AMI datasets, showing that the proposed simplified deep CNN architecture could internally perform speaker normalisation for those datasets. Both DNN and CNN hidden representations exhibited increased speaker-awarness with more abstract representations (at the middle layers) than at the input, indicating the ability of the models to disentangle speaker informa-

tion from learned representation. However, the speaker invariance curve of CNN representations was not monotonic, indicating more speaker-invariant representations at the upper layers of the network than in the middle layers. We conclude that the CNN model was able to implicitly disentangle informative speaker-related features in the middle layers, to then learn more invariant representations in the upper layers.

7.2 Limitations and future work

The main shortcoming of our work is limited generalisation of our findings to other training and testing conditions, i.e. other datasets, other NN architectures, and other frameworks. As future work, we would be the most interested to evaluate

- if embedding-based SAT-DNN could be optimised,
- if learning the representations at multiple scales would be beneficial,
- if deep CNNs could implicitly learn speaker-related differentiating features,

for other datasets, other NN architectures and other frameworks, to be able to make more general conclusions.

Different datasets and sources of variability

Evaluating the proposed methods for another dataset with similar source of variability, i.e. with reverberation and background noise, would be useful to validate our findings. E.g., using CHiME-6 [Watanabe et al., 2020] for this purpose could be interesting, as it is a difficult distant multi-microphone dataset[1]. It would also be very interesting to perform diagnostic verification for the datasets with different main sources of speaker-related speech variability (e.g. different accents, emotions, age groups, pronunciation disorders), to understand more specifically which underlying factors are informative and if deep CNNs can implicitly learn the representations which are disentangled and invariant to those factors.

Also, using some of the recently proposed data augmentation techniques such as SpecAugment [Park et al., 2019; Park et al., 2020], to evaluate the generalisation performance of the proposed methods would be an interesting research

[1]The lowest WER for the eval set in The 6th CHiME Speech Separation and Recognition Challenge (https://chimechallenge.github.io/chime6/) was 31.0%

direction. Analysing the representations learned with different augmentation techniques could provide an insight into the reasons for superior performance of one augmentation method over the other. It is not clear if embedding-based SAT-DNNs or multi-scale convolutional representations would also be beneficial for the augmentation approaches that operate directly on the input representations in time-frequency domain.

Different architectures

Evaluation of the methods with different NN architectures, such as TDNNs with different dilation factors, and Transformers could be considered as a future work. It would be interesting to evaluate learned representations for different verification tasks and compare the performance for those architectures. It would be interesting to probe the representations of a state-of-the-art Transformer-based acoustic model, e.g. a hybrid model presented by Wang et al. [2020], to better understand its learning mechanisms and the relations to CNNs and TDNNs. Also, self-attention mechanism can be seen as a way to efficiently learn at multiple scales, therefore, we would like to compare Transformers with our multi-scale CNN representations. MultiOctConv layer could also possibly be used in a decoder in end-to-end models. E.g., it could be used for positional encoding and down-sampling in a Transformer architecture. It would be interesting to determine if a Transformer model can benefit from an auxiliary embedding, or if multi-head self-attention is sufficient to implicitly learn features which change at lower rate than senone-discriminative features and correspond to other informative factors of variation.

Different frameworks

Another direction could be to probe the representations learned in a multi-task framework and compare them with the representations learned by an adversarial acoustic model. For speaker classification as the secondary task, it would be interesting to compare speaker verification layer-wise patterns between those frameworks, and relate them to WER performance, to better understand what is an optimal speaker pattern, or if there is one, and how does it depend on the type of the operations performed in the hidden layers.

Combination of proposed methods

Combinations of the proposed methods, i.e. an embedding-based SAT-CNN (with a deep all convolutional 2D CNN), and embedding-based SAT-MultiOctCNN, could bring more improvement. Since in our experiments we found that deep CNNs were successfully learning speaker differentiating features, they could benefit from incorporating an auxiliary utterance embedding capturing other sources of variability. An optimal embedding type for deep CNN acoustic models is perhaps different to an optimal embedding for a DNN or a TDNN – if the representations learned by different types of models encapsulate different type of nuisance information, then a different embedding should be complementary to such representations.

Further analysis

Analysing learned representations is a valuable research direction. It would be worth evaluating what type of information was discarded in the multi-scale representations, to better understand the reasons behind better performance when using smoothed, low resolution representations. Specifically, we are interested to test whether the assumption of smoothing is reasonable because either

1. the discarded fine-scale features corresponded to noise and the smoothed representations are a de-noised version of high resolution representations;

2. the discarded fine-scale features corresponded to the details about the senones, hence the smoothed representations are preventing the network from memorising the training data; or

3. the discarded fine-scale features corresponded to both noise and senones details.

Evaluating smoothed representations in a noise or senone recognition tasks could give some additional insight and could indicate which factors of variation are relevant for the task, and if the learned representations are disentangled. Another interesting direction could be to listen to learned representations, rather than visualising or scoring them for explanatory factors. This idea was previously investigated e.g. by Choi et al. [2015] and recently by Li et al. [2020a].

Raw waveform input

Effective and efficient deep learning from raw waveform is an active research direction. Palaz et al. [2013] showed the benefit of CNNs used in conjunction with raw waveform input for a phoneme recognition task, and Palaz et al. [2015] confirms those results for LVCSR. Sainath et al. [2015a] showed that raw waveform features match the performance of FBANKs on a large ASR task (2,000h), with a Convolutional Long Short-Term Memory Deep Neural Network (CLDNN), which uses time and frequency connvolutional layers, as well as the LSTM layers. Similarly, Ghahremani et al. [2016b] trained a CNN acoustic model directly from signal domain and show competitive performance to the models trained on conventional features with i-vector adaptation. Zhu et al. [2016] realised multi-scale representation learning with convolutional layers and show the improvements compared to spectrograms. Wav2letter [Collobert et al., 2016] is a fully-convolutional ASR system, that can be trained from raw waveform input. Zeghidour et al. [2018a] showed consistent improvement of trainable filterbanks relatively to FBANKs. The filterbank training is based on convolutions, and the acoustic model is based on wav2letter. Raw waveform input was also showed to improve children speech recognition by using a convolutional front-end in an end-to-end architecture [Dubagunta et al., 2019]. We believe that the proposed MultiOctCNN model could be successfully used for raw waveform modelling. We leave this as future work.

Better embeddings

The other possible future direction is the improvement of the design of the embeddings for the acoustic model adaptation. In this thesis, we extracted one embedding per utterance to be able to analyse the attributes at the utterance level. We hypothesise that introducing more variability to the embeddings at training time will be beneficial for the model adaptation. Using the embeddings extracted every couple of frames for training and utterance level embeddings at test time could result in a more optimal setting. Also, attention mechanism could be used in place of the pooling operation.

Embeddings for different tasks

Embeddings extracted in a way presented in this thesis can be regarded as a generic framework which is able to produce the acoustic summary vectors for sequential data. There are therefore other possible use cases for those embeddings, other than the acoustic model adaptation, for instance the selection of the augmented training data, or as a similarity measure for fMLLR initialisation. Evaluating the proposed embeddings in a speaker verification task, but in a text-dependent scenario, could also give good results, since the proposed utterance embeddings should capture senone as well speaker information. Furthermore, Williams et al. [2020] confirmed the usability of the proposed embeddings for automatic quality estimation of the utterances for multi-speaker text-to-speech synthesis. It would also be interesting to evaluate them for style transfer or voice conversion for text-to-speech synthesis.

Appendices

APPENDIX A

Multi-scale representations

A.1 Computational cost

Counting the number of computations in a model is one way of estimating its speed [Hollemans, 2018]. Usually, FLOPS (floating point operations per second) or MACCs (multiply-accumulate operations) are used in the literature for this purpose. A dot product of two vectors of length `N` uses `N` MACCs, and `2N-1` FLOPS, since there are `N` multiplications and `N-1` additions in a dot product. For simplicity, we will use MACCs in our analysis. We assume a batch size of one (a single forward pass of an input feature vector at time t), and we do not consider the cost of applying the activation function since it is negligible, especially with the ReLU activation function.

If the operation performed in a *fully-connected* layer is

$$\mathbf{y} = \mathbf{W}^T\mathbf{x} + \mathbf{b}$$

where $\mathbf{x} \in \mathbb{R}^{1xM}$, $\mathbf{y} \in \mathbb{R}^{1xN}$, the weights are stored in an `M x N` matrix $\mathbf{W}$, and bias vector $\mathbf{b} \in \mathbb{R}^{1xN}$, the total number of MACCs equals to `M x N` (a single bias addition and `N-1` additions in a dot product sum to `N` additions in total).

A *convolutional* (conv) layer operates on three-dimensional feature maps of size `H x W x C`. `H` and `W` are the spatial feature map dimensions (height and width, respectively), `C` is the number of channels. The total number of MACCs

in a convolutional layer with a square kernel of size K is therefore:

$$\text{total \#MACCs} \;=\; \texttt{K}^2 \texttt{ x Cin x Hout x Wout x Cout}$$

The padding and stride choices affect the number of MACCs – it is reflected in the dimensions of the output feature map Hout and Wout. We do not consider the cost of pooling and interpolation in the computations for convolutional layers (it is negligible compared to convolutions).

In the *Octave* and *Multi-Octave convolutional* layers, the convolution operation is factorised into multiple smaller operations (see Eq. 5.2-5.4 for an example of a MultiOctConv layer). The computational cost for each path in a MultiOctConv layer with three resolution groups is

$$\begin{aligned}
\text{\#MACCs } (Y^{1\to1}) &= \texttt{K}^2 \texttt{ x } \alpha_1\texttt{Cin x Hout x Wout x } \alpha_1\texttt{Cout} \\
\text{\#MACCs } (Y^{1\to2}) &= \texttt{K}^2 \texttt{ x } \alpha_1\texttt{Cin x Hout/2 x Wout/2 x } \alpha_2\texttt{Cout} \\
\text{\#MACCs } (Y^{1\to3}) &= \texttt{K}^2 \texttt{ x } \alpha_1\texttt{Cin x Hout/4 x Wout/4 x } \alpha_3\texttt{Cout} \\
\\
\text{\#MACCs } (Y^{2\to1}) &= \texttt{K}^2 \texttt{ x } \alpha_2\texttt{Cin x Hout/2 x Wout/2 x } \alpha_1\texttt{Cout} \\
\text{\#MACCs } (Y^{2\to2}) &= \texttt{K}^2 \texttt{ x } \alpha_2\texttt{Cin x Hout/2 x Wout/2 x } \alpha_2\texttt{Cout} \\
\text{\#MACCs } (Y^{2\to3}) &= \texttt{K}^2 \texttt{ x } \alpha_2\texttt{Cin x Hout/4 x Wout/4 x } \alpha_3\texttt{Cout} \\
\\
\text{\#MACCs } (Y^{3\to1}) &= \texttt{K}^2 \texttt{ x } \alpha_3\texttt{Cin x Hout/4 x Wout/4 x } \alpha_1\texttt{Cout} \\
\text{\#MACCs } (Y^{3\to2}) &= \texttt{K}^2 \texttt{ x } \alpha_3\texttt{Cin x Hout/4 x Wout/4 x } \alpha_2\texttt{Cout} \\
\text{\#MACCs } (Y^{3\to3}) &= \texttt{K}^2 \texttt{ x } \alpha_3\texttt{Cin x Hout/4 x Wout/4 x } \alpha_3\texttt{Cout}
\end{aligned}$$

where the α_n parameters control the number of channels associated with each resolution group n and $\sum_{n=1}^{3} \alpha_n = 1$. The computational gain comes from reduced spatial dimensions Hout and Wout of the output feature maps[1]. The total number of MACCs in such a layer is

$$\text{total \#MACCs} \;=\; \texttt{K}^2 \texttt{ x Cin x Hout x Wout x Cout x A}$$

$$A = \alpha_1^2 + \frac{1}{2}\alpha_1\alpha_2 + \frac{1}{8}\alpha_1\alpha_3 + \frac{1}{4}\alpha_2^2 + \frac{1}{8}\alpha_2\alpha_3 + \frac{1}{16}\alpha_3^2$$

We do not take into account the cost of summing the representations within the same output resolution, since this cost is negligible and we only consider

[1]Note that the convolution is performed *before* up-sampling and *after* down-sampling.

MACCs in our approximation. The total number of MACCs is proportional to $\alpha_1, \alpha_2, \alpha_3$. As an example, for [0.8, 0.1, 0.1] as $\alpha_1, \alpha_2, \alpha_3$, respectively, the scaling factor A equals 0.69 (i.e. the MultiOctConv layer uses 69% of the computations of a vanilla Conv layer). The most computationally efficient model in our experiments uses four resolution groups and 54% of vanilla computations.

A.2 Number of parameters and memory footprint

Memory bandwidth is a second aspect to consider when estimating the speed of a model. The steps for each NN layer, involving memory accesses, are [Hollemans, 2018]:

1. Reading the input representation from main memory,
2. Reading the weights[2] from main memory to perform matrix multiplications,
3. Writing the output representation to main memory.

Accesses to/from main memory are slow compared to computations. In general, since the parameters of the models are stored in main memory, the fewer the parameters, the fewer memory accesses to read the weights and the faster the model. A fully-connected layer has `M x N` weights. In a vanilla Conv layer, the parameters are shared across feature maps, so in total a Conv layer has $\texttt{K}^2$ `x Cin x Cout` weights. The number of weights for each path in a MultiOctConv layer (factorised into three sub-layers) is

$$
\begin{aligned}
\#\text{weights } (Y^{1\to1}) &= \texttt{K}^2 \;\texttt{x}\; \alpha_1\texttt{Cin} \;\texttt{x}\; \alpha_1\texttt{Cout} \\
\#\text{weights } (Y^{1\to2}) &= \texttt{K}^2 \;\texttt{x}\; \alpha_1\texttt{Cin} \;\texttt{x}\; \alpha_2\texttt{Cout} \\
\#\text{weights } (Y^{1\to3}) &= \texttt{K}^2 \;\texttt{x}\; \alpha_1\texttt{Cin} \;\texttt{x}\; \alpha_3\texttt{Cout} \\
\\
\#\text{weights } (Y^{2\to1}) &= \texttt{K}^2 \;\texttt{x}\; \alpha_2\texttt{Cin} \;\texttt{x}\; \alpha_1\texttt{Cout} \\
\#\text{weights } (Y^{2\to2}) &= \texttt{K}^2 \;\texttt{x}\; \alpha_2\texttt{Cin} \;\texttt{x}\; \alpha_2\texttt{Cout} \\
\#\text{weights } (Y^{2\to3}) &= \texttt{K}^2 \;\texttt{x}\; \alpha_2\texttt{Cin} \;\texttt{x}\; \alpha_3\texttt{Cout}
\end{aligned}
$$

[2]For simplicity, we do not consider reading the biases in the approximations.

$$
\begin{aligned}
\#\text{weights } (Y^{3\to1}) &= \texttt{K}^2 \;\texttt{x}\; \alpha_3\texttt{Cin} \;\texttt{x}\; \alpha_1\texttt{Cout} \\
\#\text{weights } (Y^{3\to2}) &= \texttt{K}^2 \;\texttt{x}\; \alpha_3\texttt{Cin} \;\texttt{x}\; \alpha_2\texttt{Cout} \\
\#\text{weights } (Y^{3\to3}) &= \texttt{K}^2 \;\texttt{x}\; \alpha_3\texttt{Cin} \;\texttt{x}\; \alpha_3\texttt{Cout}
\end{aligned}
$$

By summing the weights in all of the paths, we show that the overall number of weights in a MultiOctConv layer with any number of groups is the same as in the corresponding vanilla Conv layer, as long as $\sum_{n=1}^{N} \alpha_n = 1$ for N groups in a MultiOctConv layer:

$$
\begin{aligned}
\text{total } \#\text{weights} &= \texttt{K}^2\ \texttt{x Cin x Cout x} \\
&\qquad (\alpha_1^2 + \alpha_1\alpha_2 + \alpha_1\alpha_3 + \alpha_2\alpha_1 + \alpha_2^2 + \alpha_2\alpha_3 + \alpha_3\alpha_1 + \alpha_3\alpha_2 + \alpha_3^2) \\
&= \texttt{K}^2\ \texttt{x Cin x Cout x}\ (\alpha_1 + \alpha_2 + \alpha_3)^2 \\
&= \texttt{K}^2\ \texttt{x Cin x Cout}
\end{aligned}
$$

The number of parameters is identical, thus the memory used to read the weights is the same in MultiOctConv and Conv layers. The memory usage gains in MultiOctCNN models come from storing the input and output representations at lower resolutions, thus less accesses to the main memory are needed for reading the input and writing the output representations. Assuming that reading or writing a single value is counted as one memory access, reading the input representation in a vanilla Conv layer takes $\texttt{K}^2$ `x Cin x Hin x Win x Cout` accesses. Again, for the paths in a MultiOctConv layer with three groups, the number of memory accesses needed to read the input representations are

$$
\begin{aligned}
\#\text{input mem } (Y^{1\to1}) &= \texttt{K}^2 \;\texttt{x}\; \alpha_1\texttt{Cin} \;\texttt{x}\; \texttt{Hin} \;\texttt{x}\; \texttt{Win} \;\texttt{x}\; \alpha_1\texttt{Cout} \\
\#\text{input mem } (Y^{1\to2}) &= \texttt{K}^2 \;\texttt{x}\; \alpha_1\texttt{Cin} \;\texttt{x}\; \texttt{Hin} \;\texttt{x}\; \texttt{Win} \;\texttt{x}\; \alpha_2\texttt{Cout} \\
\#\text{input mem } (Y^{1\to3}) &= \texttt{K}^2 \;\texttt{x}\; \alpha_1\texttt{Cin} \;\texttt{x}\; \texttt{Hin} \;\texttt{x}\; \texttt{Win} \;\texttt{x}\; \alpha_3\texttt{Cout}
\end{aligned}
$$

$$
\begin{aligned}
\#\text{input mem } (Y^{2\to1}) &= \texttt{K}^2 \;\texttt{x}\; \alpha_2\texttt{Cin} \;\texttt{x}\; \texttt{Hin/2} \;\texttt{x}\; \texttt{Win/2} \;\texttt{x}\; \alpha_1\texttt{Cout} \\
\#\text{input mem } (Y^{2\to2}) &= \texttt{K}^2 \;\texttt{x}\; \alpha_2\texttt{Cin} \;\texttt{x}\; \texttt{Hin/2} \;\texttt{x}\; \texttt{Win/2} \;\texttt{x}\; \alpha_2\texttt{Cout} \\
\#\text{input mem } (Y^{2\to3}) &= \texttt{K}^2 \;\texttt{x}\; \alpha_2\texttt{Cin} \;\texttt{x}\; \texttt{Hin/2} \;\texttt{x}\; \texttt{Win/2} \;\texttt{x}\; \alpha_3\texttt{Cout}
\end{aligned}
$$

$$
\begin{aligned}
\#\text{input mem } (Y^{3\to1}) &= \texttt{K}^2 \;\texttt{x}\; \alpha_3\texttt{Cin} \;\texttt{x}\; \texttt{Hin/4} \;\texttt{x}\; \texttt{Win/4} \;\texttt{x}\; \alpha_1\texttt{Cout} \\
\#\text{input mem } (Y^{3\to2}) &= \texttt{K}^2 \;\texttt{x}\; \alpha_3\texttt{Cin} \;\texttt{x}\; \texttt{Hin/4} \;\texttt{x}\; \texttt{Win/4} \;\texttt{x}\; \alpha_2\texttt{Cout} \\
\#\text{input mem } (Y^{3\to3}) &= \texttt{K}^2 \;\texttt{x}\; \alpha_3\texttt{Cin} \;\texttt{x}\; \texttt{Hin/4} \;\texttt{x}\; \texttt{Win/4} \;\texttt{x}\; \alpha_3\texttt{Cout}
\end{aligned}
$$

The total number of accesses for reading the input representation is

$$\text{total \#input mem} \;=\; \texttt{K}^2\ \texttt{x Cin x Hin x Win x Cout x B}$$

$$B = \alpha_1^2 + \frac{5}{4}\alpha_1\alpha_2 + \frac{17}{16}\alpha_1\alpha_3 + \frac{1}{4}\alpha_2^2 + \frac{5}{16}\alpha_2\alpha_3 + \frac{1}{16}\alpha_3^2$$

and for [0.8, 0.1, 0.1] as $\alpha_1, \alpha_2, \alpha_3$, respectively, the scaling factor B equals 0.83.

Similarly, writing the output representations in a Conv layer takes `Hout x Wout x Cout` accesses, and in a MultiOctConv layer

$$\begin{aligned}
\text{\#output mem } (Y^1) &= \texttt{Hout} \;\texttt{x}\; \texttt{Wout} \;\texttt{x}\; \alpha_1\texttt{Cout} \\
\text{\#output mem } (Y^2) &= \texttt{Hout/2} \;\texttt{x}\; \texttt{Wout/2} \;\texttt{x}\; \alpha_2\texttt{Cout} \\
\text{\#output mem } (Y^3) &= \texttt{Hout/4} \;\texttt{x}\; \texttt{Wout/4} \;\texttt{x}\; \alpha_3\texttt{Cout}
\end{aligned}$$

The total number of accesses for writing the output representations is

$$\text{total \#output mem} \;=\; \texttt{Hout x Wout x Cout x C}$$

$$C = \alpha_1 + \frac{1}{4}\alpha_2 + \frac{1}{16}\alpha_3$$

and for [0.8, 0.1, 0.1] as $\alpha_1, \alpha_2, \alpha_3$, respectively, the scaling factor C equals 0.83. The memory reported in the results section corresponds to the memory used for writing the output representations. We use `float32` format, so the memory in MB is computed by multiplying the number of accesses by four bytes.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. http://tensorflow.org/. 23, 86, 130

Abdel-Hamid, O., Deng, L., and Yu, D. (2013). Exploring Convolutional Neural Network Structures and Optimization Techniques for Speech Recognition. In *Interspeech*. 33

Abdel-Hamid, O. and Jiang, H. (2013). Fast speaker adaptation of hybrid NN/HMM model for speech recognition based on discriminative learning of speaker code. In *IEEE ICASSP*. 51, 53

Abdel-Hamid, O., rahman Mohamed, A., Jiang, H., Deng, L., Penn, G., and Yu, D. (2014a). Convolutional Neural Networks for Speech Recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22. 31, 32

Abdel-Hamid, O., rahman Mohamed, A., Jiang, H., Deng, L., Penn, G., and Yu, D. (2014b). Convolutional Neural Networks for Speech Recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22. 33

Adiga, A., Magimai-Doss, M., and Seelamantula, C. S. (2013). Gammatone wavelet Cepstral Coefficients for robust speech recognition. In *TENCON*. 97

Alain, G. and Bengio, Y. (2017). Understanding intermediate layers using linear classifier probes. *arXiv*, abs/1610.01644. 127

Alam, M. J., Ouellet, P., Kenny, P., and O'Shaughnessy, D. D. (2011). Comparative Evaluation of Feature Normalization Techniques for Speaker Verification. In *NOLISP*. 167

Allebach, J. P. (2005). 7.1 Image Scanning, Sampling, and Interpolation. In *Handbook of Image and Video Processing (Second Edition)*. Academic Press. 103

Anden, J. and Mallat, S. (2014). Deep Scattering Spectrum. *IEEE Transactions on Signal Processing*, 62. 97

Anguera, X., Wooters, C., and Hernando, J. (2007). Acoustic beamforming for speaker diarization of meetings. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 15. 43

Arandjelovic, R., Gronat, P., Torii, A., Pajdla, T., and Sivic, J. (2016). NetVLAD: CNN architecture for weakly supervised place recognition. In *CVPR*. 64

Arsikere, H. and Garimella, S. (2017). Robust Online i-Vectors for Unsupervised Adaptation of DNN Acoustic Models: A Study in the Context of Digital Voice Assistants. In *Interspeech*. 66

Arsikere, H., Sapru, A., and Garimella, S. (2019). Multi-Dialect Acoustic Modeling Using Phone Mapping and Online i-Vectors. In *Interspeech*. 66

Arya, V., Bellamy, R., Chen, P., Dhurandhar, A., Hind, M., Hoffman, S. C., Houde, S., Liao, Q., Luss, R., Mojsilovic, A., Mourad, S., Pedemonte, P., Raghavendra, R., Richards, J., Sattigeri, P., Shanmugam, K., Singh, M., Varshney, K. R., Wei, D., and Zhang, Y. (2019). One Explanation Does Not Fit All: A Toolkit and Taxonomy of AI Explainability Techniques. *arXiv*, abs/1909.03012. 125

Ba, J., Kiros, J. R., and Hinton, G. E. (2016). Layer Normalization. *arXiv*, abs/1607.06450. 26, 58

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv*, abs/1409.0473. 35

Bahl, L., Brown, P., de Souza, P., and Mercer, R. (1986). Maximum mutual information estimation of hidden Markov model parameters for speech recognition. In *IEEE ICASSP*. 18

Baker, J. K. (1975). *Stochastic Modeling as a Means of Automatic Speech Recognition*. PhD thesis, Carnegie Mellon University. 2

Barker, J., Watanabe, S., Vincent, E., and Trmal, J. (2018). The Fifth âĂIJCHiMEâĂİ Speech Separation and Recognition Challenge: Dataset, Task and Baselines. In *Interspeech*. 48

Baskar, M. K., Watanabe, S., Astudillo, R., Hori, T., Burget, L., and Černocký, J. (2019). Semi-Supervised Sequence-to-Sequence ASR Using Unpaired Speech and Text. In *Interspeech*. 28

Baum, L. E. and Eagon, J. A. (1967). An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology. *Bulletin of the American Mathematical Society*, 73. 17

Belinkov, Y. and Glass, J. (2017). Analyzing Hidden Representations in End-to-End Automatic Speech Recognition Systems. In *NIPS*. 128

Bell, P., Gales, M. J., Hain, T., Kilgour, J., Lanchantin, P., Liu, X., McParland, A., Renals, S., Saz, O., Wester, M., et al. (2015). The MGB Challenge: Evaluating multi-genre broadcast media recognition. In *IEEE ASRU*. 44, 45

Bell, P. and Renals, S. (2015a). A system for automatic alignment of broadcast media captions using weighted finite-state transducers. In *IEEE ASRU*. 28

Bell, P. and Renals, S. (2015b). Regularization of context-dependent deep neural networks with context-independent multi-task training. In *IEEE ICASSP*. 52

Bell, P., Swietojanski, P., and Renals, S. (2013). Multi-level adaptive networks in tandem and hybrid ASR systems. In *IEEE ICASSP*. 53

Bellman, R. (1957). Dynamic programming. *Princeton University Press*. 10

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35. 5

Benzeghiba, M., De Mori, R., Deroo, O., Dupont, S., Erbes, T., Jouvet, D., Fissore, L., Laface, P., Mertins, A., Ris, C., Rose, R., Tyagi, V., and Wellekens, C. (2007). Automatic Speech Recognition and Speech Variability: A Review. *Speech Communication*, 49. 49

Bhattacharya, G., Alam, J., and Kenny, P. (2019). Deep Speaker Recognition: Modular or Monolithic? In *Interspeech*. 64

Bhattacharya, G., Alam, M. J., and Kenny, P. (2017). Deep Speaker Embeddings for Short-Duration Speaker Verification. In *Interspeech*. 64

Borsky, M., Mizera, P., Pollák, P., and Nouza, J. (2017). Dithering techniques in automatic recognition of speech corrupted by MP3 compression: Analysis, solutions and experiments. *Speech Communication*, 86. 12

Bourlard, H. and Morgan, N. (1989). A Continuous Speech Recognition System Embedding MLP into HMM. In *NIPS*. 2

Bourlard, H. A. and Morgan, N. (1994). *Connectionist Speech Recognition: A Hybrid Approach*. Kluwer Academic Publishers. 18

Bovik, A. C. (2009). Chapter 3 - Basic Gray Level Image Processing. In *The Essential Guide to Image Processing*. Academic Press. 104

Breiman, L. et al. (2001). Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical Science*, 16. 122

Bridle, J. (1990). Training Stochastic Model Recognition Algorithms as Networks can Lead to Maximum Mutual Information Estimation of Parameters. In *NIPS*. 22

Bridle, J. S. and Brown, M. D. (1974). An experimental automatic word recognition system. *Joint Speech Research Unit report*, 1003. 11

Bridle, J. S. and Cox, S. J. (1991). RecNorm: Simultaneous Normalisation and Classification applied to Speech Recognition. In *NIPS*. 53

Carter, S., Armstrong, Z., Schubert, L., Johnson, I., and Olah, C. (2019). Activation Atlas. *Distill*. 124

Carvalho, D. V., Pereira, E. M., and Cardoso, J. S. (2019). Machine Learning Interpretability: A Survey on Methods and Metrics. *Electronics*, 8. 123

Chan, W., Jaitly, N., Le, Q., and Vinyals, O. (2016). Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *IEEE ICASSP*. 35

Chen, C.-F., Fan, Q., Mallinar, N., Sercu, T., and Feris, R. S. (2018). Big-Little Net: An Efficient Multi-Scale Feature Representation for Visual and Speech Recognition. *arXiv*, abs/1807.03848. 97

Chen, Y., Fan, H., Xu, B., Yan, Z., Kalantidis, Y., Rohrbach, M., Yan, S., and Feng, J. (2019). Drop an Octave: Reducing Spatial Redundancy in Convolutional Neural Networks With Octave Convolution. In *ICCV*. 99, 103, 111, 117, 118

Choi, K., Kim, J., Fazekas, G., and Sandler, M. (2015). Auralisation of Deep Convolutional Neural Networks: Listening to Learned Features. In *ISMIR*. 178

Chorowski, J., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. (2015). Attention-Based Models for Speech Recognition. In *NIPS*. 35

Chorowski, J., Weiss, R. J., Bengio, S., and van den Oord, A. (2019). Unsupervised Speech Representation Learning Using WaveNet Autoencoders. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 27. 29

Choueiter, G. F. and Glass, J. R. (2007). An Implementation of Rational Wavelets and Filter Design for Phonetic Classification. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 15. 96

Chung, J. S., Huh, J., Mun, S., Lee, M., Heo, H. S., Choe, S., Ham, C., Jung, S., Lee, B.-J., and Han, I. (2020). In defence of metric learning for speaker recognition. *arXiv*, abs/2003.11982. 64

Chung, J. S., Nagrani, A., and Zisserman, A. (2018). VoxCeleb2: Deep Speaker Recognition. In *Interspeech*. 64

Chung, Y.-A. and Glass, J. (2018). Speech2Vec: A Sequence-to-Sequence Framework for Learning Word Embeddings from Speech. In *Interspeech*. 29

Collobert, R., Puhrsch, C., and Synnaeve, G. (2016). Wav2Letter: an End-to-End ConvNet-based Speech Recognition System. *arXiv*, abs/1609.03193. 179

Cordonnier, J.-B., Loukas, A., and Jaggi, M. (2020). On the Relationship between Self-Attention and Convolutional Layers. In *ICLR*. 37

Cui, X., Goel, V., and Saon, G. (2017). Embedding-Based Speaker Adaptive Training of Deep Neural Networks. In *Interspeech*. 51, 54, 71, 72

Davis, S. and Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28. 11

Dehak, N., Dehak, R., Kenny, P., Brümmer, N., Ouellet, P., and Dumouchel, P. (2009). Support vector machines versus fast scoring in the low-dimensional total variability space for speaker verification. In *Interspeech*. 60

Dehak, N., Kenny, P. J., Dehak, R., Dumouchel, P., and Ouellet, P. (2011). Front-End Factor Analysis for Speaker Verification. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 19. 60

Delcroix, M., Kinoshita, K., Hori, T., and Nakatani, T. (2015). Context adaptive deep neural networks for fast acoustic model adaptation. In *IEEE ICASSP*. 54

Delcroix, M., Kinoshita, K., Ogawa, A., Huemmer, C., and Nakatani, T. (2018). Context Adaptive Neural Network Based Acoustic Models for Rapid Adaptation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26. 54

Delcroix, M., Watanabe, S., Ogawa, A., Karita, S., and Nakatani, T. (2018). Auxiliary Feature Based Adaptation of End-to-end ASR Systems. In *Interspeech*. 58, 67

Deoras, A., Mikolov, T., and Church, K. W. (2011). A Fast Re-scoring Strategy to Capture Long-Distance Dependencies. In *EMNLP*. 20

Dong, L., Xu, S., and Xu, B. (2018). Speech-Transformer: A No-Recurrence Sequence-to-Sequence Model for Speech Recognition. In *IEEE ICASSP*. 37, 38

Doshi-Velez, F. and Kim, B. (2017). Towards A Rigorous Science of Interpretable Machine Learning. *arXiv*, abs/1702.08608. 123, 124

Drugman, T., PylkkÃűnen, J., and Kneser, R. (2016). Active and Semi-Supervised Learning in ASR: Benefits on the Acoustic and Language Models. In *Interspeech*. 27

Dubagunta, S. P., Hande Kabil, S., and Magimai-Doss, M. (2019). Improving Children Speech Recognition through Feature Learning from Raw Speech Signal. In *IEEE ICASSP*. 179

Dumoulin, V., Perez, E., Schucher, N., Strub, F., Vries, H. d., Courville, A., and Bengio, Y. (2018). Feature-wise transformations. *Distill*. 55

Dunbar, E., Algayres, R., Karadayi, J., Bernard, M., Benjumea, J., Cao, X.-N., Miskic, L., Dugrain, C., Ondel, L., Black, A. W., Besacier, L., Sakti, S., and Dupoux, E. (2019). The Zero Resource Speech Challenge 2019: TTS without T. In *Interspeech*. 29

Elloumi, Z., Besacier, L., Galibert, O., and Lecouteux, B. (2018). Analyzing Learned Representations of a Deep ASR Performance Prediction Model. In *EMNLP*. 128

Erhan, D., Bengio, Y., Courville, A. C., and Vincent, P. (2009). Visualizing Higher-Layer Features of a Deep Network. Technical report, University of Montreal. Also presented at the ICML 2009 Workshop on Learning Feature Hierarchies. 124

Farouk, M. H. (2013). *Application of Wavelets in Speech Processing*. Springer Publishing Company, Incorporated. 97

Fukuda, T., Suzuki, M., Kurata, G., Thomas, S., Cui, J., and Ramabhadran, B. (2017). Efficient Knowledge Distillation from an Ensemble of Teachers. In *Interspeech*. 126

Furui, S. (1986). Speaker-independent isolated word recognition using dynamic features of speech spectrum. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34. 13

Gabor, D. (1946). Theory of communication. Part 1: The analysis of information. *Journal of the Institution of Electrical Engineers - Part III: Radio and Communication Engineering*, 93. 96

Gales, M. and Young, S. (2007). The Application of Hidden Markov Models in Speech Recognition. *Foundations and Trends in Signal Processing*, 1. 15

Gales, M. J. F. (1998). Maximum likelihood linear transformations for HMM-based speech recognition. *Computer Speech & Language*, 12. 51, 52

Gales, M. J. F. (1999). Semi-tied covariance matrices for hidden Markov models. *IEEE Transactions on Speech and Audio Processing*, 7. 13

Garimella, S., Mandal, A., Strom, N., Hoffmeister, B., Matsoukas, S., and Parthasarathi, S. H. K. (2015). Robust i-vector based adaptation of DNN acoustic model for speech recognition. In *Interspeech*. 65

Gemello, R., Mana, F., Scanzio, S., Laface, P., and De Mori, R. (2006). Adaptation of Hybrid ANN/HMM Models Using Linear Hidden Transformations and Conservative Training. In *IEEE ICASSP*. 51, 56

Ghahremani, P., Manohar, V., Povey, D., and Khudanpur, S. (2016a). Acoustic Modelling from the Signal Domain Using CNNs. In *Interspeech*. 14

Ghahremani, P., Manohar, V., Povey, D., and Khudanpur, S. (2016b). Acoustic Modelling from the Signal Domain Using CNNs. In *Interspeech*. 179

Gill, P. (2018). *Introduction to Machine Learning Interpretability*. O'Reilly Media, Incorporated. 122

Gillick, L. and Cox, S. (1989). Some statistical issues in the comparison of speech recognition algorithms. In *IEEE ICASSP*. 11

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*. 23, 132

Golik, P., Tüske, Z., Schlüter, R., and Ney, H. (2015). Convolutional neural networks for acoustic modeling of raw time signal in LVCSR. In *Interspeech*. 14

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Nets. In *NIPS*. 59

Graves, A. (2012). Sequence Transduction with Recurrent Neural Networks. *arXiv*, abs/1211.3711. 35

Graves, A., Jaitly, N., and rahman Mohamed, A. (2013). Hybrid speech recognition with Deep Bidirectional LSTM. In *IEEE ASRU*. 34

Guo, P., Sun, S., and Xie, L. (2019). Unsupervised Adaptation with Adversarial Dropout Regularization for Robust Speech Recognition. In *Interspeech*. 59

Gupta, M. and Gilbert, A. (2001). Robust speech recognition using wavelet coefficient features. In *IEEE ASRU*. 96

Gupta, V., Kenny, P., Ouellet, P., and Stafylakis, T. (2014). I-vector-based speaker adaptation of deep neural networks for French broadcast audio transcription. In *IEEE ICASSP*. 53

Hajibabaei, M. and Dai, D. (2018). Unified Hypersphere Embedding for Speaker Recognition. *arXiv*, abs/1807.08312. 64

Hakkani-Tür, D., Riccardi, G., and Gorin, A. (2002). Active learning for automatic speech recognition. In *IEEE ICASSP*. 27

Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M., and Dally, W. (2016). EIE: Efficient Inference Engine on Compressed Deep Neural Network. *SIGARCH Comput. Archit. News*, 44. 98

Han, S., Pool, J., Tran, J., and Dally, W. (2015). Learning both Weights and Connections for Efficient Neural Network. In *NIPS*. 98

Hannun, A. Y., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., and Ng, A. Y. (2014). Deep Speech: Scaling up end-to-end speech recognition. *arXiv*, abs/1412.5567. 35

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *CVPR*. 3, 64, 97, 126, 171

Heisenberg, W. (1927). Uber den anschaulichen Inhalt der quantentheoretischen Kinematik und Mechanik. *Journal for Physics*, 43. 96

Henderson, R. and Rothe, R. (2017). Picasso: A Modular Framework for Visualizing the Learning Process of Neural Network Image Classifiers. *Journal of Open Research Software*, 5. 138

Hermann, E., Kamper, H., and Goldwater, S. (2018). Multilingual and Unsupervised Subword Modeling for Zero-Resource Languages. *arXiv*, abs/1811.04791. 29

Hinton, G., Deng, L., Yu, D., Dahl, G. E., rahman Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., and Kingsbury, B. (2012). Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine*, 29. 2, 18, 29

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the Knowledge in a Neural Network. In *NIPS*. 28, 126

Hollemans, M. (2018). How fast is my model? http://machinethink.net/blog/how-fast-is-my-model/. 183, 185

Hu, J., Shen, L., Albanie, S., Sun, G., and Wu, E. (2020). Squeeze-and-Excitation Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42. 99

Hu, K., Sak, H., and Liao, H. (2019). Adversarial Training for Multilingual Acoustic Modeling. *arXiv*, abs/1906.07093. 59

Huang, G., Liu, S., van der Maaten, L., and Weinberger, K. Q. (2017). CondenseNet: An Efficient DenseNet Using Learned Group Convolutions. In *CVPR*. 99

Huang, J.-T., Li, J., and Gong, Y. (2015). An analysis of convolutional neural networks for speech recognition. In *IEEE ICASSP*. 31, 33

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*. 25, 132

Irino, T. and Patterson, R. D. (2002). Segregating information about the size and shape of the vocal tract using a time-domain auditory model: The stabilised wavelet-Mellin transform. *Speech Communication*, 36. 97

Itoh, N., Sainath, T. N., Jiang, D. N., Zhou, J., and Ramabhadran, B. (2012). N-best entropy based data selection for acoustic modeling. In *IEEE ICASSP*. 27

Itti, L., Koch, C., and Niebur, E. (2009). A Model of Saliency-Based Visual Attention for Rapid Scene Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20. 123, 125

Jansen, A., Church, K. W., and Hermansky, H. (2010). Towards spoken term discovery at scale with zero resources. In *Interspeech*. 28

Jelinek, F. (1976). Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64. 2

Jurafsky, D. and Martin, J. H. (2009). *Speech and Language Processing (2nd Edition)*. Prentice-Hall, Inc. 12, 20, 21

Kamper, H., Jansen, A., and Goldwater, S. (2017). A segmental framework for fully-unsupervised large-vocabulary speech recognition. *Computer Speech & Language*, 46. 28

Karanasou, P., Wang, Y., Gales, M. J. F., and Woodland, P. C. (2014). Adaptation of deep neural network acoustic models using factorised i-vectors. In *Interspeech*. 66

Karita, S., Watanabe, S., Iwata, T., Delcroix, M., Ogawa, A., and Nakatani, T. (2019). Semi-supervised End-to-end Speech Recognition Using Text-to-speech and Autoencoders. In *IEEE ICASSP 2019*. 28

Karpathy, A. (2014). t-SNE visualization of CNN codes. https://cs.stanford.edu/people/karpathy/cnnembed/. 124

Kim, B., Khanna, R., and Koyejo, O. O. (2016a). Examples are not enough, learn to criticize! Criticism for Interpretability. In *NIPS*. 123

Kim, S., Raj, B., and Lane, I. (2016b). Environmental Noise Embeddings for Robust Speech Recognition. *arXiv*, abs/1601.02553. 52

Kim, T., Song, I., and Bengio, Y. (2017). Dynamic Layer Normalization for Adaptive Neural Acoustic Modeling in Speech Recognition. In *Interspeech*. 51, 58

Kindermans, P.-J., Hooker, S., Adebayo, J., Alber, M., Schütt, K. T., Dähne, S., Erhan, D., and Kim, B. (2019). The (Un)reliability of saliency methods. *arXiv*, abs/1711.00867. 125

Kindermans, P.-J., Schütt, K. T., Alber, M., Müller, K.-R., Erhan, D., Kim, B., and Dähne, S. (2018). Learning how to explain neural networks: PatternNet and PatternAttribution. In *ICLR*. 125

Kingma, D. P. and Ba, J. (2015). Adam: A Method for Stochastic Optimization. *arXiv*, abs/1412.6980. 24

Ko, T., Peddinti, V., Povey, D., Seltzer, M. L., and Khudanpur, S. (2017). A study on data augmentation of reverberant speech for robust speech recognition. In *IEEE ICASSP*. 117

Kokhlikyan, N., Miglani, V., Martin, M., Wang, E., Reynolds, J., Melnikov, A., Lunova, N., and Reblitz-Richardson, O. (2019). PyTorch Captum. https://github.com/pytorch/captum. 125

Kornblith, S., Norouzi, M., Lee, H., and Hinton, G. E. (2019). Similarity of Neural Network Representations Revisited. In *ICML*. 127

Kotnik, B., Kacic, Z., and Horvat, B. (2003). The usage of wavelet packet transformation in automatic noisy speech recognition systems. In *International Conference on Smart Technologies (EUROCON)*. 96

Krishnan Parthasarathi, S. H. and Strom, N. (2019). Lessons from Building Acoustic Models with a Million Hours of Speech. In *IEEE ICASSP*. 28

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012a). ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*. 27, 33, 34

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012b). ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*. 99

Krug, A., Knaebel, R., and Stober, S. (2018). Neuron activation profiles for interpreting convolutional speech recognition models. In *IRASL*. 128

Krug, A. and Stober, S. (2018). Introspection for convolutional automatic speech recognition. In *EMNLP*. 128

Krug, A. and Stober, S. (2020). Gradient-Adjusted Neuron Activation Profiles for Comprehensive Introspection of Convolutional Speech Recognition Models. *arXiv*, abs/2002.08125. 128

Kundu, S., Mantena, G., Qian, Y., Tan, T., Delcroix, M., and Sim, K. C. (2016). Joint acoustic factor learning for robust deep neural network based automatic speech recognition. In *IEEE ICASSP*. 66

Kurata, G. and Audhkhasi, K. (2019). Multi-Task CTC Training with Auxiliary Feature Reconstruction for End-to-End Speech Recognition. In *Interspeech*. 28

Lamel, L., Gauvain, J.-L., and Adda, G. (2002). Lightly Supervised and Unsupervised Acoustic Model Training. *Computer Speech & Language*, 16. 28

Lang, K. J., Waibel, A. H., and Hinton, G. E. (1990). A time-delay neural network architecture for isolated word recognition. *Neural networks*, 3. 29

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1989a). Handwritten Digit Recognition with a Back-Propagation Network. In *NIPS*. 33

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998a). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86. 33

LeCun, Y., Bottou, L., Orr, G., and Müller, K. (1998b). Efficient BackProp. In *Neural Networks: Tricks of the Trade*. 12

LeCun, Y., Denker, J., and Solla, S. (1989b). Optimal Brain Damage. In *NIPS*. 98

Lei, Y., Scheffer, N., Ferrer, L., and McLaren, M. (2014). A novel scheme for speaker recognition using a phonetically-aware deep neural network. In *IEEE ICASSP*. 61

Li, B. and Sim, K. C. (2010). Comparison of discriminative input and output transformations for speaker adaptation in the hybrid NN/HMM systems. In *Interspeech*. 51, 56

Li, C.-Y., Yuan, P.-C., and Lee, H.-Y. (2020a). What does a network layer hear? Analyzing hidden representations of end-to-end ASR through speech synthesis. In *IEEE ICASSP*. 178

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. (2016). Pruning Filters for Efficient ConvNets. *arXiv*, abs/1608.08710. 98

Li, J., Seltzer, M. L., Wang, X., Zhao, R., and Gong, Y. (2017). Large-Scale Domain Adaptation via Teacher-Student Learning. *arXiv*, abs/1708.05466. 28

Li, J., Wu, Y., Gaur, Y., Wang, C., Zhao, R., and Liu, S. (2020b). On the comparison of popular end-to-end models for large scale speech recognition. In *Interspeech*. 35

Li, J., Zhao, R., Huang, J.-T., and Gong, Y. (2014). Learning Small-Size DNN with Output-Distribution-Based Criteria. In *Interspeech*. 28

Liao, H. (2013). Speaker adaptation of context dependent deep neural networks. In *IEEE ICASSP*. 27, 49, 50

Liu, F.-H., Stern, R. M., Huang, X., and Acero, A. (1993). Efficient Cepstral Normalization for Robust Speech Recognition. In *HTL*. 13

Liu, T. (2020). Depth-wise Separable Convolutions: Performance Investigations. https://tlkh.dev/depsep-convs-perf-investigations/. 119

Liu, Y., Zhang, P., and Hain, T. (2014). Using neural network front-ends on far field multiple microphones based speech recognition. In *IEEE ICASSP*. 51, 53

Long, Y., Ye, H., Li, Y., and Liang, J. (2018). Active Learning for LF-MMI Trained Neural Networks in ASR. In *Interspeech*. 27

Loweimi, E., Bell, P., and Renals, S. (2019). On Learning Interpretable CNNs with Parametric Modulated Kernel-Based Filters. In *Interspeech*. 14

Lu, L., Guo, M., and Renals, S. (2017). Knowledge distillation for small-footprint highway networks. *IEEE ICASSP*. 126

Lu, L., Sun, E., and Gong, Y. (2019). Self-Teaching Networks. In *Interspeech*. 28

Lundberg, S. and Lee, S.-I. (2017). A Unified Approach to Interpreting Model Predictions. In *NIPS*. 125

Luo, J., Zhang, H., Zhou, H., Xie, C., Wu, J., and Lin, W. (2019). ThiNet: Pruning CNN Filters for a Thinner Net. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41. 99

Ma, N., Zhang, X., Zheng, H.-T., and Sun, J. (2018). ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In *ECCV*. 99

Maas, A. L., Qi, P., Xie, Z., Hannun, A. Y., Lengerich, C. T., Jurafsky, D., and Ng, A. Y. (2017). Building DNN Acoustic Models for Large Vocabulary Speech Recognition. *Computer Speech & Language*, 41. 29, 31

Mallat, S. (2012). Group Invariant Scattering. *Communications on Pure and Applied Mathematics*, 65. 97

Manohar, V., Ghahremani, P., Povey, D., and Khudanpur, S. (2018). A Teacher-Student Learning Approach for Unsupervised Domain Adaptation of Sequence-Trained ASR Models. In *IEEE SLT*. 28

Manohar, V., Povey, D., and Khudanpur, S. (2015). Semi-supervised maximum mutual information training of deep neural network acoustic models. In *Interspeech*. 28

Mei-Yuh Hwang, Xuedong Huang, and Alleva, F. A. (1996). Predicting unseen triphones with senones. *IEEE Transactions on Speech and Audio Processing*, 4. 17

Meng, Z., Li, J., Chen, Z., Zhao, Y., Mazalov, V., Gong, Y., and Juang, B. (2018). Speaker-Invariant Training Via Adversarial Learning. In *IEEE ICASSP*. 59

Meng, Z., Li, J., and Gong, Y. (2019a). Adversarial Speaker Adaptation. In *IEEE ICASSP*. 59

Meng, Z., Li, J., and Gong, Y. (2019b). Attentive Adversarial Learning for Domain-invariant Training. In *IEEE ICASSP*. 59

Miao, Y. and Metze, F. (2015). Distance-aware DNNs for robust speech recognition. In *Interspeech*. 66

Miao, Y., Zhang, H., and Metze, F. (2015). Speaker Adaptive Training of Deep Neural Network Acoustic Models Using I-Vectors. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23. 53, 60

Mitra, V. and Franco, H. (2015). Time-frequency convolutional networks for robust speech recognition. In *IEEE ASRU*. 31

Mitra, V. and Franco, H. (2018). Interpreting DNN Output Layer Activations: A Strategy to Cope with Unseen Data in Speech Recognition. In *IEEE ICASSP*. 128

Montavon, G., Lapuschkin, S., Binder, A., Samek, W., and Müller, K.-R. (2017). Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, 65. 125

Montavon, G., Samek, W., and Müller, K.-R. (2018). Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73. 122, 124, 125

Morcos, A. S., Raghu, M., and Bengio, S. (2018). Insights on representational similarity in neural networks with canonical correlation. In *NIPS*. 127

Muckenhirn, H., Abrol, V., Magimai-Doss, M., and Marcel, S. (2019). Understanding and Visualizing Raw Waveform-Based CNNs. In *Interspeech*. 128

Nagamine, T. and Mesgarani, N. (2017). Understanding the representation and computation of multilayer perceptrons: A case study in speech recognition. In *ICML*. 128

Nagamine, T., Seltzer, M. L., and Mesgarani, N. (2015). Exploring how deep neural networks form phonemic categories. In *Interspeech*. 128

Nagamine, T., Seltzer, M. L., and Mesgarani, N. (2016). On the Role of Nonlinear Transformations in Deep Neural Network Acoustic Models. In *Interspeech*. 128

Nagrani, A., Chung, J. S., Xie, W., and Zisserman, A. (2019). Voxceleb: Large-scale speaker verification in the wild. *Computer Science and Language*. 64, 166

Nagrani, A., Chung, J. S., and Zisserman, A. (2017). VoxCeleb: a large-scale speaker identification dataset. In *Interspeech*. 46, 64, 166

Nair, V. and Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. In *ICML*. 22

Neto, J., Almeida, L., Hochberg, M., Martins, C., Nunes, L., Renals, S., and Robinson, T. (1995). Speaker-adaptation for hybrid HMM-ANN continuous speech recognition system. In *Eurospeech*. 51, 52, 56

Novotney, S. and Callison-Burch, C. (2010). Cheap, Fast and Good Enough: Automatic Speech Recognition with Non-Expert Transcription. In *HLT-NAACL*. 27

Ochiai, T., Delcroix, M., Kinoshita, K., Ogawa, A., Asami, T., Katagiri, S., and Nakatani, T. (2017). Cumulative moving averaged bottleneck speaker vectors for online speaker adaptation of CNN-based acoustic models. In *IEEE ICASSP*. 55

Okabe, K., Koshinaka, T., and Shinoda, K. (2018). Attentive Statistics Pooling for Deep Speaker Embedding. In *Interspeech*. 63

Olah, C., Mordvintsev, A., and Schubert, L. (2017). Feature Visualization. *Distill*. 124

Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K., and Mordvintsev, A. (2018). The Building Blocks of Interpretability. *Distill*. 124

Palaz, D., Collobert, R., and Magimai-Doss, M. (2013). Estimating phoneme class conditional probabilities from raw speech signal using convolutional neural networks. In *Interspeech*. 179

Palaz, D., Doss, M. M., and Collobert, R. (2015a). Learning linearly separable features for speech recognition using convolutional neural networks. 126

Palaz, D., Magimai-Doss, M., and Collobert, R. (2015b). Analysis of CNN-based speech recognition system using raw speech as input. In *Interspeech*. 127

Palaz, D., Magimai-Doss, M., and Collobert, R. (2015). Convolutional Neural Networks-based continuous speech recognition using raw speech signal. In *IEEE ICASSP*. 14, 179

Palaz, D., Magimai-Doss, M., and Collobert, R. (2019). End-to-end acoustic modeling using convolutional neural networks for HMM-based automatic speech recognition. *Speech Communication*, 108. 127

Parcollet, T., Morchid, M., and LinarÃĺs, G. (2020). E2E-SINCNET: Toward Fully End-To-End Speech Recognition. In *IEEE ICASSP*. 14

Parihar, N. and Picone, J. (2002). Aurora working group: DSR front end LVCSR evaluation AU/384/02. *Technical Report*. 41

Park, A. S. and Glass, J. R. (2008). Unsupervised Pattern Discovery in Speech. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 16. 28

Park, D. S., Chan, W., Zhang, Y., Chiu, C.-C., Zoph, B., Cubuk, E. D., and Le, Q. V. (2019). SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition. In *Interspeech*. 26, 176

Park, D. S., Zhang, Y., Chiu, C., Chen, Y., Li, B., Chan, W., Le, Q. V., and Wu, Y. (2020). Specaugment on Large Scale Datasets. In *IEEE ICASSP*. 26, 176

Pascual, S., Ravanelli, M., SerrÃă, J., Bonafonte, A., and Bengio, Y. (2019). Learning Problem-Agnostic Speech Representations from Multiple Self-Supervised Tasks. In *Interspeech*. 28

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *arXiv*, abs/1912.01703. 23

Paul, D. B. and Baker, J. M. (1992). The design for the wall street journal-based CSR corpus. In *ICSLP*. 27, 41

Peddinti, V., Chen, G., Povey, D., and Khudanpur, S. (2015a). Reverberation robust acoustic modeling using i-vectors with time delay neural networks. In *Interspeech*. 65

Peddinti, V., Povey, D., and Khudanpur, S. (2015b). A time delay neural network architecture for efficient modeling of long temporal contexts. In *Interspeech*. 13, 29

Peddinti, V., Sainath, T., Maymon, S., Ramabhadran, B., Nahamoo, D., and Goel, V. (2014). Deep Scattering Spectrum with deep neural networks. In *IEEE ICASSP*. 97

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12. 140

Perez, E., Strub, F., de Vries, H., Dumoulin, V., and Courville, A. C. (2018). FiLM: Visual Reasoning with a General Conditioning Layer. *arXiv*, abs/1709.07871. 55

Pham, N.-Q., Nguyen, T.-S., Niehues, J., MÃijller, M., and Waibel, A. (2019). Very Deep Self-Attention Networks for End-to-End Speech Recognition. In *Interspeech*. 38, 39

Polyak, A. and Wolf, L. (2015). Channel-Level Acceleration of Deep Face Representations. *IEEE Access*, 3. 99

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., Silovsky, J., Stemmer, G., and Veselý, K. (2011). The Kaldi Speech Recognition Toolkit. In *IEEE ASRU*. 26, 85, 129

Povey, D., Hadian, H., Ghahremani, P., Li, K., and Khudanpur, S. (2018). A Time-Restricted Self-Attention Layer for ASR. In *IEEE ICASSP*. 37, 38

Povey, D., Peddinti, V., Galvez, D., Ghahremani, P., Manohar, V., Na, X., Wang, Y., and Khudanpur, S. (2016). Purely Sequence-Trained Neural Networks for ASR Based on Lattice-Free MMI. In *Interspeech*. 19, 35

Povey, D. and Woodland, P. C. (2002). Minimum Phone Error and I-smoothing for improved discriminative training. In *IEEE ICASSP*. 18

Povey, D., Zhang, X., and Khudanpur, S. (2015). Parallel training of DNNs with Natural Gradient and Parameter Averaging. In *ICLR*. 78

Qian, Y., Bi, M., Tan, T., and Yu, K. (2016). Very deep convolutional neural networks for noise robust speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24. 133

Qian, Y. and Woodland, P. C. (2016). Very deep convolutional neural networks for robust speech recognition. In *IEEE SLT*. 98

Qian, Y. and Woodland, P. C. (2017). Very deep convolutional neural networks for robust speech recognition. In *IEEE SLT*. 3, 32, 33, 34, 129, 130, 133

Qian, Y., Yin, M., You, Y., and Yu, K. (2015). Multi-task joint-learning of deep neural networks for robust speech recognition. In *IEEE ASRU*. 52

Rabiner, L. R., Juang, B., Levinson, S. E., and Sondhi, M. M. (1985). Recognition of isolated digits using hidden Markov models with continuous mixture densities. *AT&T Technical Journal*, 64. 18

Raghu, M., Gilmer, J., Yosinski, J., and Sohl-Dickstein, J. (2017). SVCCA: Singular Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability. In *NIPS*. 126

rahman Mohamed, A., Dahl, G., and Hinton, G. (2009). Deep belief networks for phone recognition. In *NIPS*. 28

rahman Mohamed, A., Hinton, G., and Penn, G. (2012). Understanding how Deep Belief Networks perform acoustic modelling. In *IEEE ICASSP*. 127

rahman Mohamed, A., Okhonko, D., and Zettlemoyer, L. (2019). Transformers with convolutional context for ASR. *arXiv*, abs/1904.11660. 37, 38

Raj, D., Snyder, D., Povey, D., and Khudanpur, S. (2019). Probing the Information Encoded in X-Vectors. In *IEEE ASRU*. 127

Ranjan, R., Castillo, C. D., and Chellappa, R. (2017). L2-constrained Softmax Loss for Discriminative Face Verification. *arXiv*, abs/1703.09507. 65

Ravanelli, M. and Bengio, Y. (2018). Speaker Recognition from Raw Waveform with SincNet. In *IEEE SLT*. 14

Ravanelli, M., Parcollet, T., and Bengio, Y. (2019). The PyTorch-Kaldi Speech Recognition Toolkit. In *IEEE ICASSP*. 86

Ravanelli, M., Zhong, J., Pascual, S., Swietojanski, P., Monteiro, J., Trmal, J., and Bengio, Y. (2020). Multi-Task Self-Supervised Learning for Robust Speech Recognition. In *IEEE ICASSP*. 28

Renals, S. (2019). Lecture 16. End-to-end systems 2: Encoder-decoder models. ASR 2018-19 course material, http://www.inf.ed.ac.uk/teaching/courses/asr/index-2019.html. 35

Renals, S., Hain, T., and Bourlard, H. (2007). Recognition and understanding of meetings: The AMI and AMIDA projects. In *IEEE ASRU*. 43

Renkens, V. (2016). Kaldi with TensorFlow Neural Net. https://github.com/vrenkens/tfkaldi. 130

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). Why Should I Trust You?: Explaining the Predictions of Any Classifier. In *ACM SIGKDD*. 125

Riccardi, G. and Hakkani-Tür, D. (2005). Active learning: theory and applications to automatic speech recognition. *IEEE Transactions on Speech and Audio Processing*, 13. 27

Robinson, T., Hochberg, M., and Renals, S. (1996). The Use of Recurrent Neural Networks in Continuous Speech Recognition. In *Automatic Speech and Speaker Recognition: Advanced Topics*, pages 233–258. Springer US. 34

Rownicka, J., Bell, P., and Renals, S. (2018). Analyzing Deep CNN-Based Utterance Embeddings for Acoustic Model Adaptation. In *IEEE SLT*. 47, 67, 121

Rownicka, J., Bell, P., and Renals, S. (2020). Multi-scale Octave Convolutions for Robust Speech Recognition. In *IEEE ICASSP*. 95, 121

Rownicka, J., Renals, S., and Bell, P. (2017). Simplifying very deep convolutional neural network architectures for robust speech recognition. In *IEEE ASRU*. 47, 67, 68, 86, 106, 121, 129

Rownicka, J., Renals, S., and Bell, P. (2019). Embeddings for DNN speaker adaptive training. In *IEEE ASRU*. 47, 67, 121

Sainath, T., Weiss, R. J., Wilson, K., Senior, A. W., and Vinyals, O. (2015a). Learning the Speech Front-end with Raw Waveform CLDNNs. In *Interspeech*. 179

Sainath, T. N., Kingsbury, B., Mohamed, A., Dahl, G. E., Saon, G., Soltau, H., Beran, T., Aravkin, A. Y., and Ramabhadran, B. (2013a). Improvements to deep convolutional neural networks for LVCSR. In *IEEE ASRU*. 31

Sainath, T. N., Kingsbury, B., Saon, G., Soltau, H., rahman Mohamed, A., Dahl, G., and Ramabhadran, B. (2015b). Deep Convolutional Neural Networks for Large-scale Speech Tasks. *Neural Networks*, 64. 31, 130

Sainath, T. N., Mohamed, A.-r., Kingsbury, B., and Ramabhadran, B. (2013b). Deep convolutional neural networks for LVCSR. In *IEEE ICASSP*. 31, 32

Sainath, T. N., Weiss, R. J., Senior, A. W., Wilson, K. W., and Vinyals, O. (2015c). Learning the speech front-end with raw waveform CLDNNs. In *Interspeech*. 14

Samarakoon, L. and Sim, K. C. (2016a). Factorized Hidden Layer Adaptation for Deep Neural Network Based Acoustic Modeling. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24. 51, 57, 60

Samarakoon, L. and Sim, K. C. (2016b). On combining i-vectors and discriminative adaptation methods for unsupervised speaker normalization in DNN acoustic models. In *IEEE ICASSP*. 60

Saon, G., Kurata, G., Sercu, T., Audhkhasi, K., Thomas, S., Dimitriadis, D., Cui, X., Ramabhadran, B., Picheny, M., Lim, L.-L., Roomi, B., and Hall, P. (2017). English Conversational Telephone Speech Recognition by Humans and Machines. In *Interspeech*. 2, 48

Saon, G., Soltau, H., Nahamoo, D., and Picheny, M. (2013). Speaker adaptation of neural network acoustic models using i-vectors. In *IEEE ASRU*. 51, 53, 65

Sarı, L., Thomas, S., and Hasegawa-Johnson, M. A. (2019). Learning Speaker Aware Offsets for Speaker Adaptation of Neural Networks. In *Interspeech*. 58

Schneider, S., Baevski, A., Collobert, R., and Auli, M. (2019). wav2vec: Unsupervised Pre-Training for Speech Recognition. In *Interspeech*. 28

Schwartz, R., Chow, Y., Kimball, O., Roucos, S., Krasner, M., and Makhoul, J. (1985). Context-dependent modeling for acoustic-phonetic recognition of continuous speech. In *IEEE ICASSP*. 17

Seide, F., Li, G., Chen, X., and Yu, D. (2011). Feature engineering in Context-Dependent Deep Neural Networks for conversational speech transcription. In *IEEE ASRU*. 51, 52

Seltzer, M. L. and Droppo, J. (2013). Multi-task learning in deep neural networks for improved phoneme recognition. In *IEEE ICASSP*. 51, 52

Seltzer, M. L., Yu, D., and Wang, Y. (2013). An investigation of deep neural networks for noise robust speech recognition. In *IEEE ICASSP*. 51, 53, 66

Senior, A. and Lopez-Moreno, I. (2014). Improving DNN speaker independence with I-vector inputs. In *IEEE ICASSP*. 53, 65

Sercu, T. and Goel, V. (2016a). Advances in Very Deep Convolutional Neural Networks for LVCSR. In *Interspeech*. 3, 32, 33, 129

Sercu, T. and Goel, V. (2016b). Advances in Very Deep Convolutional Neural Networks for LVCSR. In *Interspeech*. 98

Sercu, T., Puhrsch, C., Kingsbury, B., and LeCun, Y. (2016a). Very deep multilingual convolutional neural networks for LVCSR. In *IEEE ICASSP*. 33, 129

Sercu, T., Puhrsch, C., Kingsbury, B., and LeCun, Y. (2016b). Very deep multilingual convolutional neural networks for LVCSR. In *IEEE ICASSP*. 98

Serdyuk, D., Audhkhasi, K., Brakel, P., Ramabhadran, B., Thomas, S., and Bengio, Y. (2016). Invariant Representations for Noisy Speech Recognition. *arXiv*, abs/1612.01928. 59

Sermanet, P., Kavukcuoglu, K., Chintala, S., and LeCun, Y. (2013). Pedestrian detection with unsupervised multi-stage feature learning. In *CVPR*. 33

Shinohara, Y. (2016). Adversarial Multi-Task Learning of Deep Neural Networks for Robust Speech Recognition. In *Interspeech*. 51, 59

Shon, S., Tang, H., and Glass, J. R. (2018). Frame-Level Speaker Embeddings for Text-Independent Speaker Recognition and Analysis of End-to-End Model. In *IEEE SLT*. 63

Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *arXiv*, abs/1312.6034. 124, 125

Simonyan, K. and Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. In *ICLR*. 3, 33, 34, 64, 98

Singh, K., Okhonko, D., Liu, J., Wang, Y., Zhang, F. F., Girshick, R. B., Edunov, S., Peng, F., Saraf, Y., Zweig, G., and rahman Mohamed, A. (2020). Training ASR Models By Generation of Contextual Information. In *IEEE ICASSP*. 28

Snyder, D., Chen, G., and Povey, D. (2015). MUSAN: A Music, Speech, and Noise Corpus. *arXiv*, abs/1510.08484. 117

Snyder, D., Garcia-Romero, D., and Povey, D. (2015). Time delay deep neural network-based universal background models for speaker recognition. In *IEEE ASRU*. 61

Snyder, D., Garcia-Romero, D., Povey, D., and Khudanpur, S. (2017). Deep Neural Network Embeddings for Text-Independent Speaker Verification. In *Interspeech*. 62

Snyder, D., Garcia-Romero, D., Sell, G., Povey, D., and Khudanpur, S. (2018). X-vectors: Robust DNN Embeddings for Speaker Recognition. In *IEEE ICASSP*. 63

Snyder, D., Garcia-Romero, D., Sell, G., Povey, D., and Khudanpur, S. (2018). X-Vectors: Robust DNN Embeddings for Speaker Recognition. In *IEEE ICASSP*. 166

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. A. (2015). Striving for Simplicity: The All Convolutional Net. In *ICLR*. 32, 126, 171

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15. 25

Stafylakis, T., Rohdin, J., Plchot, O., Mizera, P., and Burget, L. (2019). Self-Supervised Speaker Embeddings. In *Interspeech*. 28

Sun, S., Yeh, C.-F., Hwang, M., Ostendorf, M., and Xie, L. (2018). Domain Adversarial Training for Accented Speech Recognition. In *IEEE ICASSP*. 59

Sundararajan, M., Taly, A., and Yan, Q. (2017). Axiomatic Attribution for Deep Networks. In *ICML*. 125

Swietojanski, P., Bell, P., and Renals, S. (2015). Structured output layer with auxiliary targets for context-dependent acoustic modelling. In *Interspeech*. 52

Swietojanski, P., Ghoshal, A., and Renals, S. (2013). Hybrid acoustic models for distant and multichannel large vocabulary speech recognition. In *IEEE ASRU*. 44

Swietojanski, P., Ghoshal, A., and Renals, S. (2014). Convolutional neural networks for distant speech recognition. *IEEE Signal Processing Letters*, 21. 31, 32

Swietojanski, P., Li, J., and Renals, S. (2016). Learning hidden unit contributions for unsupervised acoustic model adaptation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24. 57, 60

Swietojanski, P. and Renals, S. (2014). Learning hidden unit contributions for unsupervised speaker adaptation of neural network acoustic models. In *IEEE SLT*. 51, 57

Swietojanski, P. and Renals, S. (2016). SAT-LHUC: Speaker adaptive training for learning hidden unit contributions. In *IEEE ICASSP*. 51, 57

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *CVPR*. 3, 33, 34, 99, 126, 171

Tan, M. and Le, Q. V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *ICML*. 99

Tan, T., Qian, Y., Hu, H., Zhou, Y., Ding, W., and Yu, K. (2018). Adaptive very deep convolutional residual network for noise robust speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26. 34

Tan, T., Qian, Y., Hu, H., Zhou, Y., Ding, W., and Yu, K. (2018). Adaptive Very Deep Convolutional Residual Network for Noise Robust Speech Recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26. 34, 56, 98, 129, 133

Tan, T., Qian, Y., Yin, M., Zhuang, Y., and Yu, K. (2015). Cluster adaptive training for deep neural network. In *IEEE ICASSP*. 51, 56

Tan, T., Qian, Y., and Yu, K. (2016). Cluster Adaptive Training for Deep Neural Network Based Acoustic Model. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24. 56

Tenney, I., Xia, P., Chen, B., Wang, A., Poliak, A., McCoy, R. T., Kim, N., Durme, B. V., Bowman, S. R., Das, D., and Pavlick, E. (2019). What do you learn from context? Probing for sentence structure in contextualized word representations. *arXiv*, abs/1905.06316. 145

Thomas, S., Seltzer, M. L., Church, K., and Hermansky, H. (2013). Deep neural network features and semi-supervised training for low resource speech recognition. In *IEEE ICASSP*. 28

Thompson, J., Bengio, Y., and Schoenwiesner, M. (2019). The effect of task and training on intermediate representations in convolutional neural networks revealed with modified RV similarity analysis. In *Conference on Cognitive Computational Neuroscience (CCN)*. 127

Tòth, L. (2014). Combining time- and frequency-domain convolution in convolutional neural network-based phone recognition. In *IEEE ICASSP*. 33

Tòth, L. (2017). Multi-resolution spectral input for convolutional neural network-based speech recognition. In *International Conference on Speech Technology and Human-Computer Dialogue (SpeD)*. 97

Tran, D. T., Delcroix, M., Ogawa, A., Huemmer, C., and Nakatani, T. (2017). Feedback connection for deep neural network-based acoustic modeling. In *IEEE ICASSP*. 67

Tsunoo, E., Kashiwagi, Y., Kumakura, T., and Watanabe, S. (2019). Transformer ASR with Contextual Block Processing. In *IEEE ASRU*. 38

Uber (2020). A model-agnostic visual debugging tool for machine learning. https://github.com/uber/manifold. 125

van der Maaten, L. (2013). Barnes-Hut-SNE. *arXiv*, abs/1301.3342. 140

van der Maaten, L. and Hinton, G. (2008). Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9. 124

Variani, E., Lei, X., McDermott, E., Moreno, I. L., and Gonzalez-Dominguez, J. (2014). Deep neural networks for small footprint text-dependent speaker verification. In *IEEE ICASSP*. 62

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is All you Need. *arXiv*, abs/1706.03762. 35, 37, 38

Veselý, K., Hannemann, M., and Burget, L. (2013). Semi-supervised training of Deep Neural Networks. In *IEEE ASRU*. 28

Veselý, K., Segura, C., Szöke, I., Luque, J., and Černocký, J. (2018). Lightly Supervised vs. Semi-supervised Training of Acoustic Model on Luxembourgish for Low-resource Automatic Speech Recognition. In *Interspeech*. 28

Veselý, K., Watanabe, S., ŽmolÃŋková, K., Karafiát, M., Burget, L., and Černocký, J. H. (2016). Sequence summarizing neural network for speaker adaptation. In *IEEE ICASSP*. 51, 58, 67, 69

Vestman, V. and Kinnunen, T. (2018). Supervector Compression Strategies to Speed up I-Vector System Development. *arXiv*, abs/1805.01156. 61

Viikki, O. and Laurila, K. (1998). Cepstral Domain Segmental Feature Vector Normalization for Noise Robust Speech Recognition. *Speech Communication*, 25. 13

Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13. 17

von Platen, P., Zhang, C., and Woodland, P. (2019). Multi-Span Acoustic Modelling Using Raw Waveform Signals. In *Interspeech*. 97

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. J. (1989). Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37. 29

Wang, Y., rahman Mohamed, A., Le, D., Liu, C., Xiao, A., Mahadeokar, J., Huang, H., Tjandra, A., Zhang, X., Zhang, F. F., Fuegen, C., Zweig, G., and Seltzer, M. L. (2020). Transformer-Based Acoustic Modeling for Hybrid Speech Recognition. In *IEEE ICASSP*. 38, 39, 177

Watanabe, S., Delcroix, M., Metze, F., and Hershey, J. R., editors (2017). *New Era for Robust Speech Recognition, Exploiting Deep Learning*. Springer. 48, 50, 51, 56, 66

Watanabe, S., Hori, T., Le Roux, J., and Hershey, J. R. (2017). Student-teacher network learning with enhanced features. In *IEEE ICASSP*. 126

Watanabe, S., Hori, T., Roux, J. L., and Hershey, J. R. (2017). Student-teacher network learning with enhanced features. In *IEEE ICASSP*. 28

Watanabe, S., Mandel, M., Barker, J., and Vincent, E. (2020). CHiME-6 Challenge: Tackling Multispeaker Speech Recognition for Unsegmented Recordings. *arXiv*, abs/2004.09249. 176

Wattenberg, M., ViÃľgas, F., and Johnson, I. (2016). How to Use t-SNE Effectively. *Distill*. 124

Weninger, F., AndrÃľs-Ferrer, J., Li, X., and Zhan, P. (2019). Listen, Attend, Spell and Adapt: Speaker Adapted Sequence-to-Sequence ASR. In *Interspeech*. 52

Williams, J., Rownicka, J., Oplustil, P., and King, S. (2020). Comparison of Speech Representations for Automatic Quality Estimation in Multi-Speaker Text-to-Speech Synthesis. In *Odyssey*. 180

Wong, J. H. and Gales, M. J. (2016). Sequence Student-Teacher Training of Deep Neural Networks. In *Interspeech*. 28

Woodland, P. and Povey, D. (2002). Large scale discriminative training of hidden Markov models for speech recognition. *Computer Speech & Language*, 16. 18

Woodland, P. C., Liu, X., Qian, Y., Zhang, C., Gales, M. J. F., Karanasou, P., Lanchantin, P., and Wang, L. (2015). Cambridge university transcription systems for the multi-genre broadcast challenge. In *IEEE ASRU*. 2

Wu, C. and Gales, M. J. F. (2015). Multi-basis adaptive neural network for rapid adaptation in speech recognition. In *IEEE ICASSP*. 56

Xie, W., Nagrani, A., Chung, J. S., and Zisserman, A. (2019). Utterance-level aggregation for speaker recognition in the wild. In *IEEE ICASSP*. 64

Xiong, W., Droppo, J., Huang, X., Seide, F., Seltzer, M., Stolcke, A., Yu, D., and Zweig, G. (2016). Achieving Human Parity in Conversational Speech Recognition. *arXiv*, abs/1610.05256. 34, 48

Xiong, W., Wu, L., Alleva, F., Droppo, J., Huang, X., and Stolcke, A. (2017). The Microsoft 2017 Conversational Speech Recognition System. In *IEEE ICASSP*. 33, 34

Xue, S., Abdel-Hamid, O., Jiang, H., Dai, L., and Liu, Q. (2014). Fast Adaptation of Deep Neural Network Based on Discriminant Codes for Speech Recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22. 53

Yang, X., Wang, K., and Shamma, S. A. (1992). Auditory representations of acoustic signals. *IEEE Transactions on Information Theory*, 38. 97

Yeh, C.-F., Mahadeokar, J., Kalgaonkar, K., Wang, Y., Le, D., Jain, M., Schubert, K., Fuegen, C., and Seltzer, M. L. (2019). Transformer-Transducer: End-to-End Speech Recognition with Self-Attention. *arXiv*, abs/1910.12977. 37, 38

Yoo, S., Song, I., and Bengio, Y. (2019). A Highly Adaptive Acoustic Model for Accurate Multi-dialect Speech Recognition. In *IEEE ICASSP*. 55, 59

Yosinski, J., Clune, J., Nguyen, A. M., Fuchs, T. J., and Lipson, H. (2015). Understanding Neural Networks Through Deep Visualization. *arXiv*, abs/1506.06579. 124

Young, S. J. and Woodland, P. C. (1994). State clustering in hidden Markov model-based continuous speech recognition. *Computer Speech & Language*, 8. 17

Yu, D. and Li, J. (2017). Recent progresses in deep learning based acoustic models. *IEEE/CAA Journal of Automatica Sinica*, 4. 34, 48, 49

Yu, D., Varadarajan, B., Deng, L., and Acero, A. (2009). Active Learning and Semi-supervised Learning for Speech Recognition: A Unified Framework using the Global Entropy Reduction Maximization Criterion. *Computer Speech & Language*. 27

Yu, D., Xiong, W., Droppo, J., Stolcke, A., Ye, G., Li, J., and Zweig, G. (2016a). Deep Convolutional Neural Networks with Layer-wise Context Expansion and Attention. In *Interspeech*. 32, 33, 129

Yu, D., Xiong, W., Droppo, J., Stolcke, A., Ye, G., Li, J., and Zweig, G. (2016b). Deep Convolutional Neural Networks with Layer-wise Context Expansion and Attention. In *Interspeech*. 33, 98

Yu, D., Yao, K., Su, H., Li, G., and Seide, F. (2013). KL-Divergence Regularized Deep Neural Network Adaptation For Improved Large Vocabulary Speech Recognition. In *IEEE ICASSP*. 51

Zeghidour, N., Usunier, N., Synnaeve, G., Collobert, R., and Dupoux, E. (2018a). End-to-End Speech Recognition From the Raw Waveform. In *Interspeech*. 179

Zeghidour, N., Xu, Q., Liptchinsky, V., Usunier, N., Synnaeve, G., and Collobert, R. (2018b). Fully Convolutional Speech Recognition. *arXiv*, abs/1812.06864. 3

Zeiler, M. D. and Fergus, R. (2014). Visualizing and Understanding Convolutional Networks. In *ECCV*. 124, 126

Zhang, P., Liu, Y., and Hain, T. (2014). Semi-supervised DNN training in meeting recognition. In *IEEE SLT*. 28

Zhao, T., Zhao, Y., and Chen, X. (2015). Time-frequency kernel-based CNN for speech recognition. In *Interspeech*. 31

Zhao, Y., Li, J., Zhang, S.-X., Chen, L., and Gong, Y. (2018). Domain and Speaker Adaptation for Cortana Speech Recognition. In *IEEE ICASSP*. 54

Zhong, Y., Arandjelovic, R., and Zisserman, A. (2018). GhostVLAD for set-based face recognition. *arXiv*, abs/1810.09951. 64

Zhou, T., Zhao, Y., Li, J., Gong, Y., and Wu, J. (2019). CNN with Phonetic Attention for Text-Independent Speaker Verification. In *IEEE ASRU*. 65

Zhu, Y., Ko, T., Snyder, D., Mak, B., and Povey, D. (2018). Self-Attentive Speaker Embeddings for Text-Independent Speaker Verification. In *Interspeech*. 63

Zhu, Z., Engel, J., and Hannun, A. Y. (2016). Learning Multiscale Features Directly from Waveforms. In *Interspeech*. 179