AUTOMATIC SPEECH RECOGNITION ENGINES ADAPTED FOR EMBEDDED PLATFORMS

Amrutha Prasad

Idiap-Com-01-2020

AUGUST 2020

Automatic Speech Recognition Engines Adapted for Embedded Platforms

Master AI THESIS

Author :	Amrutha Prasad
Student n° :	18-696-500
Project supervisor :	Petr Motlicek
Company supervisor :	Alexandre Nanchen

Abstract

Conventional hybrid automatic speech recognition (ASR) engines exploit state-of-the-art Deep Neural Network (DNN) based acoustic model (AM) trained with Lattice Free-Maximum Mutual Information (LF-MMI) criterion and n-gram Language Model (LM). These systems usually have a large number of parameters and therefore require significant parameter reduction to operate on embedded devices.

This thesis studies an impact of the parameter reduction on the overall speech recognition performance. Following three approaches are presented: (i) AM trained in the Kaldi framework with conventional factorized TDNN (TDNN-F) architecture. (ii) the TDNN built in Kaldi is loaded into the Pytorch toolkit using a C++ wrapper. The weights and activation parameters are then quantized and the inference is performed in Pytorch. (iii) post quantization training for fine-tuning. Results obtained on standard Librispeech setup provide an interesting overview of recognition accuracy with respect to applied quantization schemes.

Furthermore, this thesis revisits Keyword Spotting (KWS) approaches and demonstrates that LF-MMI AM built to classify context-independent phones can operate well when integrated within a light-weight decoder providing a likelihood ratio based confidence score. The KWS was compared with the conventional lattice-based system on several keyword detection datasets.

Acknowledgements

This work was supported by the CTI Project “SHAPED: Speech Hybrid Analytics Platform for consumer and Enterprise Devices". I would like to thank Dr. Petr Motlicek and Alexandre Nanchen for their dedicated supervision. Their help and assistance during this thesis provided me with invaluable insights in this subject. I also like to thank Dr. Srikanth Madikeri for providing the support with the implementation of the Kaldi C++ wrapper in Pytorch. I wish to acknowledge Arash Salarian for providing valuable insights and suggestions regarding quantization.

Martigny, June 17, 2020 A. P.

Contents

List of Figures

List of Tables

1 Introduction

The goal of the Speech Hybrid Analytics Platform for Consumer and Enterprise Devices (SHAPED) project[1] is to develop a software architecture and related algorithms to enable the most effective processing of speech between the embedded device and the cloud, balance user experience and cost of the operation across the range of Logitech voice-enabled interface devices.

With evolving technology, companies aim to provide its user effortless usability of their devices. This can be achieved through implementing voice-control ability in their devices, allowing the user to control the device using the most natural and obvious means of communication (e.g. speech). The vision is to embed certain functionalities including turning the device on/off, playing music requested by the user and perform most of these operations on-device avoiding the use of cloud-processing.

For instance, devices from Logitech that currently use cloud-computing to perform the operations are offered by the Alexa (i.e. Amazon cloud). The illustration of a Smart device that uses cloud-computation is shown in Figure 1.1. This functionality raises many issues that define the goals of the SHAPED project. Users that intend to use speech to interact with devices are required to be registered with the cloud service. In other words, Logitech products need to rely on third-party services. In addition, cloud service providers require ownership of processed data, which implies that the company cannot exploit the customer data for further analytics or processing to improve their services. Finally, uploading the data to the cloud raises ethical questions related to sensitivity of the processed data of the clients. Clients use the service with the consent that their information will not be shared or stored.

The above mentioned concerns motivated the project consortium to implement the Automatic Speech Recognition (ASR) and Keyword Spotting (KWS) functionalities on

[1]https://www.idiap.ch/en/scientific-research/projects/SHAPED

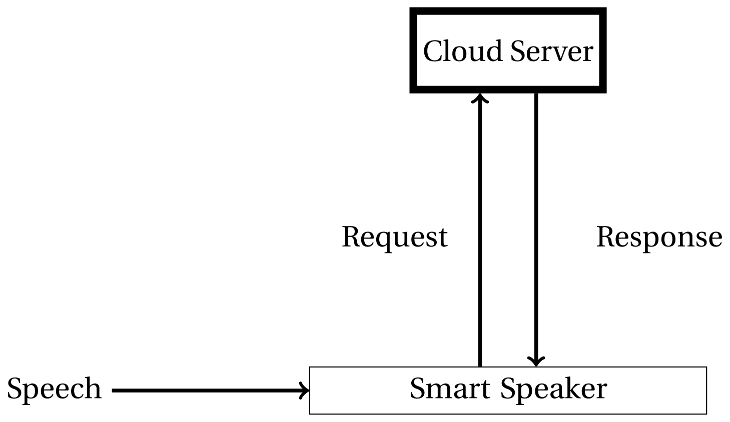

Figure 1.1 – Illustration of cloud computation in Smart devices.

the embedded device thus minimizing the use of cloud-computation. To achieve this we propose to reduce the model size for ASR through different approaches, allowing to implement the full ASR/KWS systems on embedded devices.

Deep Neural Networks (DNN) help learn multiple levels of representation of data in order to model complex relationships among them. Conventional Acoustic Model (AM) applied in ASR framework is usually trained with neural network architectures such as Convolutional Neural Networks (CNN) [1], Recurrent Neural Networks (RNNs) [2], Time-delay Neural Networks (TDNN) [3] with the Kaldi [4] toolkit. The models with such architecture can be complex and each layer usually has millions of parameters. As the number of layers increases, the parameters of the model increase as well, which has a direct impact on the AM footprint. Such models with millions of parameters make them impractical to use with embedded devices such as Raspberry Pi. To embed ASR/KWS systems on such devices, the footprint of the system needs to be significantly reduced. One simple solution is to train a model with fewer parameters. However, reducing the model size usually decreases the performance of the system.

Conventional KWS systems trained with Kaldi toolkit are based on word-recognition lattices generated by the kaldi decoder using a large FST graph (i.e. usually of size of hundreds of MB). As this approach is not practical for embedded devices, we started to consider several alternative approaches to significantly reduce the CPU and memory requirements.

One of the often considered approach is to quantize the model parameters from floating point values to integers. In [5], quantization methods are studied for CNN architectures for image classification and other computer vision problems. Results show that quantizing such models reduces the model size significantly without any impact on the performance. Another approach is to use teacher-student training to first train a larger model that is optimized for performance and use its output to train a smaller model. Alternatively, in models such as [6] parameter reduction is integrated as a part of training. In this thesis, we study the effect of quantizing the parameters of an AM used in ASR with a focus on deploying it on embedded devices with low computational resources (especially, memory). We present the impact on the performance of the ASR system when the AM is quantized from "float32" to int8 or int16. The results of the quantization process are then compared to other techniques used in parameter reduction for automatic speech recognition models. We believe that results obtained from our study have not been presented in literature so far, and can be of interest for researchers experimenting with interfacing Kaldi and Pytorch [7] tools for ASR tasks.

Specifically for KWS engines, we also started to consider to significantly reduce the CPU requirements (and also requirements for memory allocation during decoding) by replacing a conventional LVCSR (lattice-based) keyword search by an acoustic based KWS [8]. The acoustic KWS does not require a heavy FST-based decoder, but can be replaced by a simple network (generated in an HTK style) to run a simple token-passing decoder (i.e. SLratio available from STK toolkit [9])[2]. In order to reach high detection accuracies, we learn from the previous work on flat-start LF-MMI TDNN modeling that context-independent models can provide as good performance as context-dependent (triphone) ones. This solution was therefore integrated into the acoustic KWS, where TDNN model (trained to classify context-independent phones) generates pseudo-likelihoods to be used as observation probabilities in the decoder (i.e. graph is built from word-based HMM topology).

Part of this thesis is submitted as a paper [10] to Interspeech 2020. The rest of the thesis is organized as follows. Chapter 2 provides an overview of the current state-of-the-art ASR system, current KWS system and briefly describes the current techniques used in parameter reduction of a model. This is followed by Chapter 3 that explains the factorized TDNN used in training of AM for parameter reduction and Chapter 4 presents an overview of the quantization techniques and their application to AM training with Kaldi toolkit. The experiments and the results are presented in Chapter 5. Finally, the conclusion is provided in Chapter 6.

[2]https://speech.fit.vutbr.cz/en/software/hmm-toolkit-stk

2 Related work

2.1 Overview of the state-of-the-art ASR

Automatic Speech Recognition (ASR) is a subfield of speech processing that involves converting speech, typically in one language, to text. Hence, this is also termed as speech-to-text. To convert a speech signal to text, a typical ASR system employs an AM and a LM. The former is trained with a set of speech recordings with their corresponding (ideally manually corrected) text, which are also referred to as transcripts. AM represents the relationship between a speech signal and the phonemes, or other linguistic units that make up the speech. The latter is trained on a large corpus of text data. LM is usually represented by a probability distribution over sequences of words. The LM provides context to distinguish between words and phrases that sound similar. Using the knowledge of AM and LM, a decoding graph is usually built as a finite state transducer (FST) [11; 12; 13], which generates text output given an observation sequence as shown in Figure 2.1.

To build a robust speech recognition engine on smaller devices, the artificial intelligence behind it has to be better at handling challenges such as different acoustic conditions, background noise, model size, performance. Developments in natural language processing and neural network technology have improved speech and voice technology in the past. The state-of-the-art hybrid system for ASR uses deep neural networks (DNN) based AMs trained with Lattice Free-Maximum Mutual Information (LF-MMI) [14] criterion and n-gram models for LM. Then, RNN based LMs are often used for rescoring the lattices generated with n-gram models. Although the lattice can be generated in real-time, rescoring with RNN-LMs is slower than real-time. This may not be suitable for applications where latency is critical.

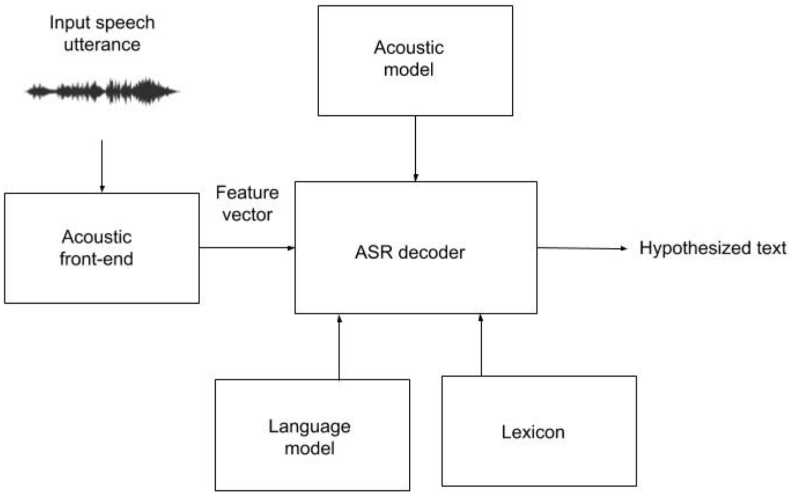

Figure 2.1 – A typical ASR system.

2.1.1 Acoustic Models

In hybrid[1] DNN-Hidden Markov Model (HMM) based AM used in the ASR system higher-level features and concepts are defined in terms of lower-level ones, and such a hierarchy of features defines the deep architecture. Most of these models are based on unsupervised learning of representations. Early works such as Feedforward Neural Networks (FNN) only take current time steps as input. RNNs, especially Long Short-Term Memory (LSTM) [15; 16] networks, have demonstrated better results in speech recognition tasks due to their cyclic connections and utilization of sequential information. However, the training of RNN relies on backpropagation, that arise issues related to time consumption, gradient vanishing and exploding due to its complex computation. CNNs that apply local connectivity, weight sharing, and pooling techniques, have outperformed previous models [17; 18].

The HMM is a generative model usually used with a Gaussian mixture model (GMM), or a DNN to model acoustic data. A common approach for learning the HMM parameters is through maximum likelihood (ML) estimation with the objective function defined as:

[1]In hybrid systems, DNNs are used to estimate probabilities for acoustic units (produced by clustering of the context) which corresponds to the HMM states. These probabilities are used to compute the likelihood of acoustic data given word sequence in a hidden Markov model (HMM).

$$F_{ML}(\lambda) = \sum_u \log p_\lambda(X_u|W_u), \tag{2.1}$$

where λ is the parameters of the AM, X_u is the u^{th} acoustic observation with transcription W_u.

In information theory, the Mutual Information (MI) of two random variables is a measure of the mutual dependence between the two variables. More specifically, it quantifies the amount of information obtained about one random variable through observing the other random variable. MI is linked to entropy, which quantifies the expected amount of information held in a random variable. In ASR, MI is measured between the distributions of observations and word sequences. Maximum Mutual Information (MMI) estimation defined as:

$$\begin{aligned} F_{MMIE}(\lambda) &= \sum_u \log \frac{p(X_u, W_u)}{p(X_u)P(W_u)} \\ &= \sum_u \left(\log \frac{p(X_u, W_u)}{p(X_u)} - \log P(W_u) \right) \\ &= \sum_u \left(\log \frac{p_\lambda(X_u, W_u)^K P(W_u)}{\sum_W p_\lambda(X_u|W)^K P(W)} - \log P(W_u) \right), \end{aligned} \tag{2.2}$$

aims to directly maximise the posterior probability that is also called as conditional maximum likelihood which is defined as:

$$\begin{aligned} F_{CML}(\lambda) &= \sum_u \log P(W_u|X_u) \\ &= \sum_u \log \frac{p_\lambda(X_u|W_u)^K P(W_u)}{\sum_W p_\lambda(X_u|W)^K P(W)}. \end{aligned} \tag{2.3}$$

The notations are described in Table 2.1.

Maximum Likelihood Estimation (MLE) is a method of estimating the parameters of a statistical model, given observations. The method obtains the parameter estimates by finding the parameter values that maximizes the likelihood function. Minimizing the cross entropy minimizes the expected frame error and maximizes the MI which reduces the sentence error thus achieving state-of-the-art performance.

The denominator in equation 2.3 is called the denominator graph and is estimated using a phone language model (LM) (instead of a word LM) as shown in [14]. The

Table 2.1 – Notations for equations 2.2 and 2.3

Notation	Definition
λ	Parameters of the acoustic model
u	Index over utterances
X	Sequence of acoustic observations
X_u	Observation for utterance u
W	Sequence of words
W_u	Transcription of $u^{th} utterance$
K	Scaling factor for correction of overestimation. = 1/12

phone LM for the denominator graph is a pruned n-gram LM trained using the phone alignments of the training data.

To increase the performance of the ASR system with the increase in the availability of resources, DNNs are trained to output pseudo likelihoods for the states in the HMM. Traditional DNN-HMM hybrid AMs are trained with the Cross Entropy (CE) criterion. Speech recognition can be considered as a sequence-to-sequence mapping problem where a sequence of sounds is converted to a sequence of meaningful linguistic units (e.g. phones, syllables, words, etc). In order to better distinguish different classes of sounds, it is useful to train AM with positive and negative examples. Hence, sequential discriminative criteria such as MMI and state-level Maximum Bayes Risk (sMBR) can be applied. The former is now commonly known as Lattice-Free MMI (LF-MMI/Chain model) [14]. This method could be used without any CE initialization leading to lesser computation [18].

State-level sequence-discriminative training of DNNs starts from a set of alignments and lattices that are generated by decoding the training data with a LM. For each training condition, the alignments and lattices are generated using the corresponding DNN trained using cross-entropy [18]. The cross-entropy trained models are also used as the starting point for the sequence-discriminative training. In sMBR training word-level LM is used, whereas in LF-MMI training phone-level LM is used. This simplification enables LF-MMI training to use GPU clusters and is considered as the state-of-the-art AM for ASR hybrid systems. However, sMBR training is 100 times slower than real-time as it can be run only on the CPU.

2.1.2 Language Models

Typical LM used in an ASR system is a statistical n-gram model which is represented by the probability distribution over a sequence of words with n being 1 (unigram), 2 (bi-gram), 3 (tri-gram), or higher. The n-gram models predict a word given the previous n-1 words by calculating the conditional probability of a word given n-1 previous words. In an n-gram LM, the probability $P(w_1,...,w_N)$ of observing the sentence $w_1,...,w_N$ is approximated as:

$$\begin{aligned} P(w_1,...,w_N) &= \prod_{i=1}^{N} P(w_i|w_1,..w_{i-1}) \\ &\approx \prod_{i=1}^{N} P(w_i|w_{i-(n-1)},..w_{i-1}), \end{aligned} \tag{2.4}$$

where n is the order of the model. Using the Markov property, in equation 2.4, the i^{th} word w_i is estimated as the probability of observing its preceding $n-1$ words. In ASR systems, a typical LM trained is a tri-gram (3-gram) model. For example, in 3-gram model, the probability of the sentence "This is a sample sentence" is approximated as:

$$\begin{aligned} P(\text{This, is, a, sample, sentence}) = {} & P(\text{This}|\text{<s>, <s>}) \times P(\text{is}|\text{this, <s>}) \\ & \times P(\text{a}|\text{is, this}) \times P(\text{sample}|\text{a, is}) P(\text{sentence}|\text{sample, a}) \\ & \times P(\text{</s>}|\text{sentence, sample}), \end{aligned} \tag{2.5}$$

where $< s >$ is start-of-sentence marker and $< /s >$ is end-of-sentence marker. To improve the performance of these LMs, they have to be trained on large amounts of data and 4-gram or 5-gram models should be used. As n increases, the size of the model and training time increase as well. Moreover, with such large LMs it is not feasible to build a decoding graph. With increase in quantity of data, the performance of the n-gram models may also decrease [19].

Research in natural language processing [20] has shown that it is possible to train neural LMs that perform better than n-gram models [21]. The possibility of training LMs with neural networks has been investigated since many years. The first neural network based LM was proposed by Bengio [21] in 2003. The statistical LMs were replaced by neural network based LMs with the commencement of RNN based LM [22]. As RNNs use their internal state to process the sequence of inputs, the training of LMs for quite large sequences of words can be achieved easily.

In contrast to the feedforward neural network, a RNN has recurrent or cyclic connections between its layers. There are connections between the input and the hidden layers which can be parameterized by a weight matrix U, connections between hidden layers can be parameterized by a weight matrix W and the connections between the hidden layers and the output can be parameterized by a weight matrix V. Many different architectures for RNNs exist. A notable difference between feedforward and recurrent networks is the way in which parameters are shared between different parts of the model. Parameter sharing allows a model to be extended to examples of different lengths and to generalize across examples [22] and is especially important when the same piece of information occurs at several positions within an input sequence. When training a machine learning model to extract certain information, we would like the model to recognize it independent of its position in the sentence. A feedforward network that processes sentences of fixed length would have different parameters for each input feature, causing it to learn separately the rules of the language at each sentence. Different to a feedforward network, an RNN would share the same weights across multiple time steps [22]. The sharing of parameters arises from the way in which an RNN operates: when computing the output of a hidden unit, each component of the output is produced by applying the same update rule to each component of the previous output. Furthermore, the model contains an output layer that uses information from the hidden state in order to make predictions. As the state of a hidden neuron at time step t is a function of all inputs from previous time steps, the recurrent connections can be viewed as creating a kind of "memory". A hidden unit that preserves information across multiple time steps is called a memory cell. Long Short-Term Memory (LSTM) is one of them which is designed to be better at accumulating information and learning long term dependencies.

During the training of an RNN LM, the input sequence of words is mapped to its embedding vector using the model's embedding matrix which can be initiated randomly and adapted during training or provided with a pre-trained word embeddings. The sequence of word embeddings is then processed by one or more hidden layers of the network. The final outputs are further processed by an output layer with a softmax activation function. For each input word, the output layer produces a vector with as many entries as words in the vocabulary, where each entry represents the predicted probability that the corresponding word is the next word in the sequence. The network is trained by minimizing CE loss between the predicted sequence and the target sequence.

Due to their simplicity, statistical n-gram models were dominant for a long time. However, RNN LMs have several advantages compared to n-gram models. RNN LMs can handle longer histories of inputs than n-gram models and are able to incorporate long-range dependencies in their predictions. This is especially important for NLP because dependencies between words often span across several sentences. RNN LM models require a lot less memory space compared to the statistical n-gram models. While n-gram models count the occurrences of certain word combinations, RNNs learn a distributed representation for each word context. This distribution can be used to obtain a probability distribution over all words. To use a large size n-gram model in ASR they have to be pruned which is not required in RNN LMs.

The performance of a language model is measured in terms of perplexity. It is a measurement of how well a model predicts a sample. As mentioned in [20], perplexity is the inverse of the probability assigned to the test set by the language model, normalized by the number of words in the test set. i.e., if a test set has $W = \{w_1, ..w_N\}$ words, then the perplexity is estimated as:

$$PP(W) = \sqrt[N]{\frac{1}{PP(w_1, w_2, ..w_N)}}. \tag{2.6}$$

If a model can assign high probability to an unseen word, then its probability is good.

Using equation 2.4 in the equation 2.6, we get

$$PP(W) = \sqrt[N]{\frac{1}{\prod_{i=1}^{N} PP(w_i | w_{i-(n-1)}, ..w_{i-1})}}. \tag{2.7}$$

2.2 Overview of the KWS system

In speech processing, acoustic keyword spotting is the task of identifying certain keywords in speech utterances. A KWS system takes a speech utterance (while a phoneme sequence of the keyword is also of interest to build a specific decoding graph) and provides a prediction of the keyword in the speech utterance as a confidence score. If the confidence is above a predefined threshold, the keyword is detected, otherwise the keyword is not accepted. Unlike ASR, KWS is a limited vocabulary system and does not recognize the whole utterance. This system is used in applications such as information retrieval, wake-word detection in smart devices. An advantage of KWS system is the possibility to spot out-of-vocabulary words, which usually cannot be recognized by Large Vocabulary Continuous Speech Recognition (LVCSR) systems.

2 types of KWS are popular and often considered by researchers but also developers: (i) a lattice based KWS [23] and (ii) an acoustic KWS [24; 8]. The lattice-based KWS is built on top of the LVCSR recognition system. It uses word recognition lattices generated by the decoder (i.e. in our experiments Kaldi based FST decoder is used to detect a keyword.

More specifically, the Kaldi KWS baseline deploys the word-recognition lattices generated from all the speech utterances that are converted from individual weighted finite state transducers (WFST) to a single generalized factor transducer structure in which the lattice posterior probability of each word is stored. This factor transducer

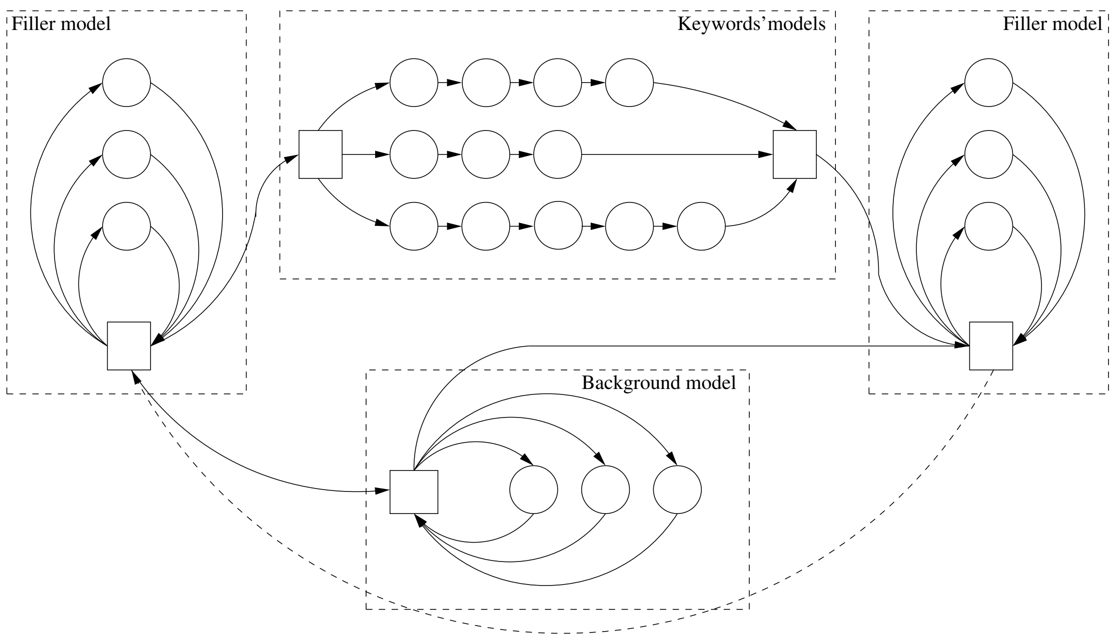

Figure 2.2 – Overview of acoustic KWS.

is actually an inverted index of all word sequences seen in the lattices [23]. During detection, given a keyword, a simple finite state machine is created that accepts the keyword and composes it with the factor transducer to obtain all occurrences of the keyword in the search collection, along with the lattice posterior probability of each occurrence. All those occurrences are sorted according to their posterior probabilities and then a decision is assigned to each instance as proposed in [25].

In acoustic KWS, during detection word models of searched keywords are built from corresponding phone models (i.e., usually 3-state phone posterior estimates are transformed into 3-state HMM with emission probabilities given by the DNN (for instance trained with LF-MMI criterion)). In parallel, concatenated keyword models are then accompanied by filler and background models (usually represented by simple phone loops to allow any possible sequence of phonemes) to create a final decoding network, as shown in Figure 2.2 . The filler model is used to represent other parts of acoustic sequence appearing in verified hypotheses (i.e. keyword is present in utterance) and the background model is used to represent the competing hypothesis (i.e. keyword is not present in utterance). Likelihoods of the detected keywords are taken from the last state of each keyword model (i.e. verified hypothesis) and from the background model (competing hypothesis) and the likelihood ratio is assigned with each keyword. The likelihoods are computed using Viterbi decoder). Confidence Score (CS) of each detected keyword is therefore given as a log-likelihood ratio between these two likelihoods [24]. As this type of KWS does not require generation of recognition

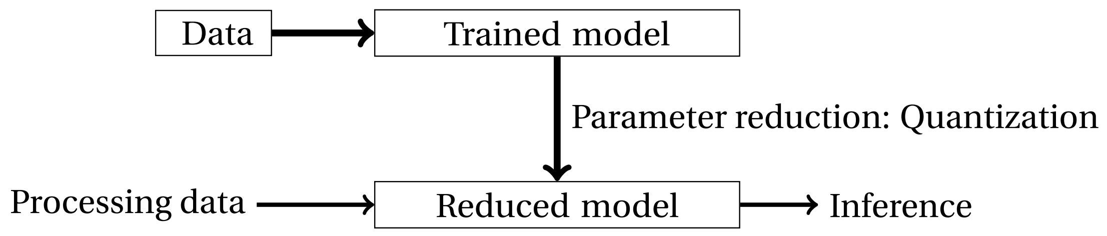

Figure 2.3 – Illustration of parameter reduction.

lattices, it offers a much lighter solution (i.e. requiring very small CPU load and memory footprint).

2.3 Overview of parameter reduction

Parameter reduction is a process that removes certain layers of the neural network avoiding the loss of useful information of the network required for its decision process. This process can be applied to already trained neural networks or implemented during the training. This process is illustrated in Figure 2.3. Several different approaches can be considered:

- Teacher-student approach to reduce the number of layers in the student neural network [26; 27].
- Reduce the size of the layers used in training the neural network through matrix factorization [28].
- Reduce the hidden layer dimension (e.g. from 1024 to 512 in each layer of the neural network).
- Reduce the number of hidden layers used in the network.
- Quantization of model parameters (e.g. from 32 bit floating precision to 16 bit floating precision) [5; 29].

2.3.1 Teacher-student approach

A simple way to improve the performance of a machine learning algorithm is to train ensemble models with the same data and then average their predictions or train a single large model. Unfortunately, the ensemble models or a large model cannot be employed on devices which have low resources. Caruana et al. in [30] showed that the knowledge in an ensemble can be compressed into a single model. This approach was

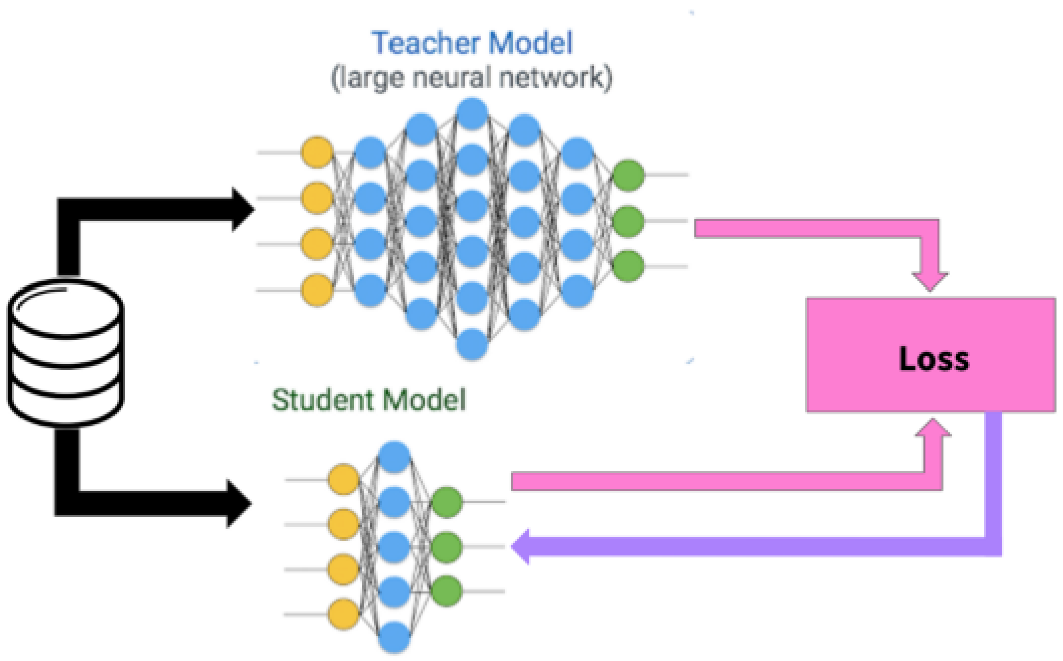

Figure 2.4 – Diagram of the teacher-student approach to reduce the model of the student network.

developed further by Hinton and his collaborators [26] through a different approach. In general, model compression in teacher-student approach is achieved by teaching a smaller network to mimic a bigger trained model. This process of applying the knowledge learnt from a larger (teacher) model to a small (student) model is known as knowledge distillation. In the distillation process, knowledge is transferred from the teacher model to the student by minimizing a loss function where the targets are the soft targets predicted by the teacher model. This process is represented in Figure 2.4.

Most often, the output of the last layer of neural network architecture for a classifier uses a softmax function. The output of this softmax function is a probability distribution that has a very high probability for the correct class, and a probability close to zero for other classes. Given the inputs z_i, the output of the softmax (σ)is estimated as:

$$\sigma(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)}. \tag{2.8}$$

In [30], the soft targets were the inputs to the final softmax (logits) while in [26] the soft targets were the output of the softmax function. The concept of "softmax temperature" was introduced by Hinton where, the probability p_i of class i is calculated from the logits z as

$$p_i = \frac{exp(\frac{z_i}{T})}{\sum_j exp(\frac{z_j}{T})}, \tag{2.9}$$

where T is the temperature parameter. When $T = 1$ the standard softmax function is obtained. As T increases, the probability distribution generated by the softmax function becomes softer. This provides more information as to which classes the teacher found more similar to the predicted class. Therefore in the distillation process, this embedded knowledge in the teacher model, will be transferred to the student model. The same higher value of T is used to compute the softmax on the student's logits during the training.

In addition to learning of the soft targets, learning to produce the correct labels based on the ground truth is found beneficial while training the small model. Hence, the "standard" loss between the student's predicted class probabilities and the ground-truth labels (also called "hard targets") is also calculated using the temperature of $T = 1$.

2.3.2 Low rank matrix factorization

Single Value Decomposition (SVD) is one of the most popular methods which can be applied to the trained models to factorize the learned weight matrix as a product of two much smaller factors. SVD then discards the smaller singular values followed by fine tuning of the network parameters to obtain a parameter-reduced model [31]. For a given $m \times n$ weight matrix W, SVD is applied as

$$W_{m\times n} = U_{m\times m} \sum_{m\times n} V^T_{n\times n}, \tag{2.10}$$

where $\sum$ is a diagonal matrix that contains the singular values of W, and U and V contains the left and right singular values of W.

3 Factorized TDNN

Speech can be viewed as a sequence-to-sequence mapping problem. Neural network architectures such as CNN and RNN were used to incorporate long temporal contexts. An important factor to be considered during the training of large neural networks is the parallelization of training so that we can use GPUs. One of the architectures effective for modeling temporal context dependencies is TDNN [32] architecture. [3] showed that the TDNN architecture does not impose any relationship between the length of the input context and the number of sequential steps used during training. This allows to model training with long temporal contexts.

3.1 Overview of TDNN

In TDNN architecture the initial transform learns from a narrow context and the higher layers learn from a wider temporal context. Hence, the deeper layers are able to learn wider temporal relationships. Each layer in the TDNN architecture uses a different temporal resolution. The resolution increases as the higher layers of the network are reached. TDNN is also called as 1D CNN because the transforms in the TDNN model are tied only across time steps. Due to this tying, the lower layers of the network learn translation invariant feature transforms during back-propagation as they are updated by a gradient accumulated over all the time steps of the input temporal context [32].

At each layer, certain time steps defined by the input context are used to compute the activations, i.e. the input contexts of each layer of the network are required to compute output activation at one time step shown in Figure 3.1. The input layer consists of $t-10$ to $t+10$ frames. The first layer has a context of $[-1,1]$ with $n=256$ where n is the dimension of each time frame. The context $[-1,1]$ implies that the t^{th} frame of layer 1 is computed using $\{t-1, t, t+1\}$ frames of the input layer. For instance,

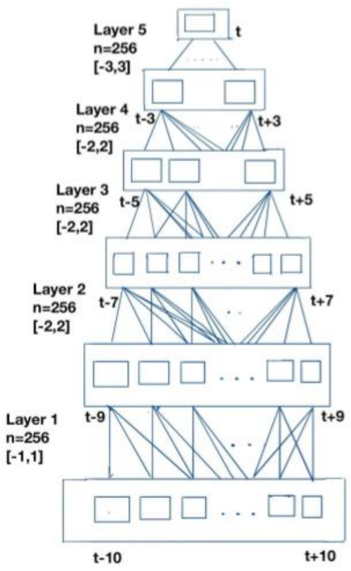

Figure 3.1 – Block diagram of computation of the output activations for each layer using the context in TDNN.

the first layer's $(t-9)^{th}$ frame is calculated using the input layer's $(t-10)^{th}$ frame to $(t-8)^{th}$ frame. However, there would be large overlaps of the input contexts when computing the activation of neighbouring time steps. Therefore using the assumption that the neighbouring activations will be correlated, sub-sampling is performed on the layers.

Sub-sampling allows gaps between frames, i.e instead of using $\{t-1, t, t+1\}$ input frames to compute the activation, it uses only $\{t-1, t+1\}$ frames. This means that selective time steps need to be evaluated. The subsampling is usually done for higher layers with the difference between the offsets at the hidden layers being multiples of 3 in order to ensure that only a small number of activations will be evaluated. With this scheme, the necessary computation is reduced during the forward pass and backpropagation. Computation reduction in forward pass and backpropagation implies that the training time is reduced. The current state-of-the-art systems use rectified linear unit (ReLU) activation followed by batchnorm layer after the TDNN layer as it provides better performance.

3.2 Overview of TDNN-F

As described in Chapter 2, SVD is one of the most popular techniques that can be applied to a model to reduce its parameters. Another approach to enforce parameter reduction while training a neural network AM is by applying low-rank factorized layers [6]. In semi-orthogonal factorization, the parameter matrix M is factorized as a product of two matrices A and B, i.e $M = AB$ where B is a semi-orthogonal matrix and A has a smaller “interior” (i.e. rank) than that of M. The term "semi-orthogonal" refers to orthogonality of non-square matrices. i.e., a non square matrix M is semi-orthogonal if $MM^T = I$ or $M^T M = I$. This can be shown using SVD. Let the parameter matrix M has fewer rows than columns. In order for M to be orthogonal, it should satisfy $MM^T = I$. Since, we try to enforce a non-square matrix M to be orthogonal, $MM^T = P$ is defined and the error is represented as $Q = P - I$. In order to minimize the error Q, a function f is defined as the sum of squared elements of Q which is represented as:

$$f = tr(QQ^T). \tag{3.1}$$

Since Q is a matrix, the sum of squared elements is $tr(QQ^T)$. f can be minimized by taking the derivative with respect to Q. Since f in equation 3.1 will be a scalar, derivative of a scalar w.r.t. a matrix is not transposed w.r.t. that matrix (i.e. QQ^T will be considered as Q^2). Hence

$$\frac{\partial f}{\partial Q} = 2Q. \tag{3.2}$$

Since Q is related to P, the derivative of f w.r.t. P is:

$$\begin{aligned}\frac{\partial f}{\partial P} &= \frac{\partial Q}{\partial P}\\ &= \frac{\partial[(P-I)(P-I)^T]}{\partial P}\\ &= \frac{\partial[PP^T - PI^T - IP^T + II^T]}{\partial P}\\ &= 2P - I^T - I\\ &= 2(P-I)\\ &= 2Q.\end{aligned} \tag{3.3}$$

Also, since we want to enforce M to be semi-orthogonal, the function f has to be minimized w.r.t. to M. The derivative of f w.r.t. to M is the value that would be used to update the value of the parameter M during one iteration of stochastic gradient descent (SGD). A detailed derivation is as below:

$$\begin{aligned}\frac{\partial f}{\partial M} &= \frac{\partial Q}{\partial M} \\ &= \frac{\partial[(P-I)(P-I)^T]}{\partial M} \\ &= \frac{\partial[PP^T - PI^T - IP^T + II^T]}{\partial M} \\ &= \frac{\partial[(MM^T)(MM^T)^T - (MM^T)I^T - I(MM^T)^T + II^T]}{\partial M} \\ &= 2M(MM^T) + (MM^T)2M - 2MI^T - 2MI \\ &= 4M(MM^T - I) \\ &= 4MQ. \end{aligned} \tag{3.4}$$

From the above derivation, we see that M is updated as $M = M - 4\nu QM$, where ν is the learning rate. For quadratic convergence, the learning rate ν is chosen to be $1/8$. By substituting the value of learning rate and simplifying we get

$$\begin{aligned} M &\leftarrow M - 4(\frac{1}{8})QM \\ M &\leftarrow M - \frac{1}{2}(MM^T - I)M. \end{aligned} \tag{3.5}$$

If M is very far from being orthonormal, the above equation would not converge. Hence, in practice Glorot (also known as Xavier) initialization [33] is used in this architecture where the standard deviation of the random initial elements of M is the inverse of square root of the number of columns.

In general, l_2 regularization is used (i) to control how fast the parameters of various layers change, and (ii) to reduce the scale of the parameters which helps the model to learn faster. Since the ReLU is scale invariant, l_2 regularization does not have an effect on how fast the parameters change when applied to the hidden layers. In order to have a consistent control of this, a scaling factor is added to equation 3.5. Substituting $\frac{1}{\alpha}M$ and simplifying equation 3.5, a scaled version of the semi-orthogonal matrix is

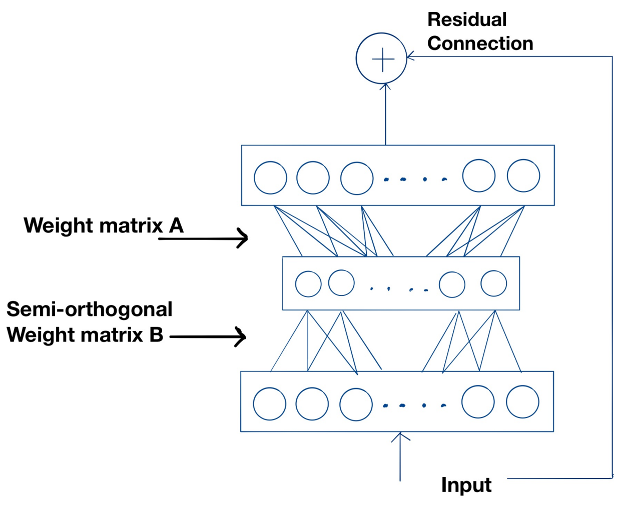

Figure 3.2 – Block diagram of computation of the output activations for one layer factorized TDNN (TDNN-F).

obtained which is

$$M \leftarrow M - \frac{1}{2\alpha^2}(MM^T - \alpha^2 I)M. \tag{3.6}$$

The formula for α that ensures that the change in M is orthogonal to itself is given as:

$$\alpha = \sqrt{\frac{tr(PP^T)}{tr(P)}}. \tag{3.7}$$

The above proposed method is applied to an existing topology such as TDNN and factorizes its parameter matrices into products of two smaller matrices. For example, if the TDNN model architecture has a hidden layer dimension of 625, then a typical parameter matrix would have a dimension of 625 × 1875. Here the number of columns (1875) corresponds to the three frame offsets of previous layers spliced together i.e., it can be viewed as a CNN with a 3 × 1 kernel and 625 filters. So the idea of factorization is to factorize this 625 × 1875 matrix M into two matrices as $M = AB$ with a smaller "interior" dimension (also called linear bottleneck dimension) of 160 for e.g. This factorization will make A of size 625 × 160 and B of size 160 × 1875 with B constrained to be semi-orthogonal. This is shown in Figure 3.2. The number of parameters

$625 * 1875 = 11.7M$ in TDNN model is reduced to $625 * 160 + 160 * 1875 = 4M$ in the TDNN-F model.

Thus this technique enables training a smaller network from scratch instead of using a pre-trained network for parameter reduction. The LF-MMI training also provides a stable training procedure for semi-orthogonalized matrices.

While semi-orthogonal matrices have been studied with TDNN-F (a variant of TDNN with residual connections), it has not been compared with other model reduction techniques. In our experiments, we present

- Comparison of TDNN and TDNN-F with respect to varying the number of layers for the ASR task.
- Comparison of using TDNN-F and different quantization techniques applied to TDNN model in monophone and triphone based neural network outputs for ASR task.
- Comparison of using monophone and triphone based neural network outputs with TDNN-F architecture for KWS task.

4 Quantization

A popular technique to reduce the size of the model is through quantization. This approach is applied in computer vision problems and is supported by many deep learning frameworks like Pytorch [7] and TensorFlow [34]. However, applying quantization to AM trained with LF-MMI criterion using Kaldi toolkit is not a straightforward approach because Kaldi inference supports only floating point operations. The following subsections explain the standard quantization process for DNNs and how it is applied to the AMs.

4.1 Overview of quantization process

Quantization is a process of mapping a set of real valued inputs to a discrete valued output. Commonly used quantization types are "16 bit" and "8 bit" (signed/unsigned) integers. Quantizing model parameters typically decreases the number of bits used to represent the parameters. Prior to this process the model may have been trained with IEEE "float32" or "float64". A model size can be reduced by a factor of 2 (with 16 bits quantization) and by a factor of 4 (with 8 bits quantization) if the original model uses "float32" representation.

In addition to the quantization types, there are different quantization modes such as symmetric and asymmetric quantization. As mentioned earlier, a real valued variable x in the range of (x_{min}, x_{max}) is quantized to a range $(qmin, qmax)$. In symmetric quantization, the range $(qmin, qmax)$ corresponds to $(\frac{-N_{levels}}{2}, \frac{N_{levels}}{2} - 1)$. In asymmetric quantization the quantization range is $(0, \frac{N_{levels}}{2} - 1)$. In the aforementioned intervals, $N_{levels} = 2^{16} = 65536$ for 16 bit quantization and $N_{levels} = 2^{8} = 256$ for 8 bit quantization.

A real value r, can be expressed as an integer q given a scale S and zero-point Z [5]:

$$r = S * (q - Z). \tag{4.1}$$

In the above equation, scale S specifies the step size required to map the floating point to integer and an integer zero-point represents the floating point zero [5].

Given the minimum and maximum of a vector x and the range of the quantization scheme, scale and zero-point is computed as below [29]:

$$S = \frac{x_{max} - x_{min}}{qmax - qmin} \tag{4.2}$$

and

$$Z = qmin - \frac{x_{min}}{scale}. \tag{4.3}$$

As mentioned in [5], for 8 bit integer quantization the values never reach -128 and hence we use $qmin = -127$ and $qmax = 127$.

4.2 Implementation

We implement the quantization algorithms in Pytorch as it provides better support than Kaldi for int8, uint8 and int16 types. The aim of our work is to port models trained in Kaldi to be functional in embedded systems. There already exist tools such as Pykaldi [35] that help users to load Kaldi acoustic models for inference in Pytorch. However, they do not allow access to the model parameters by default. To support this work, we implemented a C++ wrapper that allows access to the model parameters and input MFCC features as Pytorch tensors. These tensors are then quantized using the above equations. The inference is then carried out in Pytorch or Kaldi, based on the type of quantization. In order to carry out inference in Kaldi, the Pytorch tensors are converted back to the format required by Kaldi. The wrapper allows us to write the models and ark (archive) files back to Kaldi format.

Once the model is loaded as a tensor, there exist several options: we can quantize only the weights of the models, or quantize both weights and activations.

4.3 Quantization of weights only

Weight-only quantization is an approach in which only the weights of the neural network model are quantized. This approach is useful when only the model size

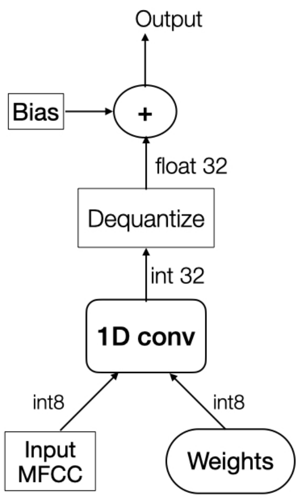

Figure 4.1 – Block diagram of integer arithmetic inference with quantization of weights and activations. Input activations and weights are represented as 8-bit integer according to equation 1. The 1D convolution involves integer inputs and a 32-bit integer accumulator. The output of the convolution is mapped back to floating point and added with the bias.

needs to be reduced and the inference is carried out in floating-point precision. In our experiments, the weights are quantized in Pytorch and the inference is carried out in Kaldi. The quantized weights cannot directly be used in Kaldi as it supports only floating-point precision. Therefore the model parameters are de-quantized (convert integer values back to floating point precision) and saved as Kaldi nnet3 model. The decoding is then performed as usual using the HCLG graph in Kaldi [4; 12].

4.4 Quantization of weights and activations

In order to reduce the model from 32 bit precision to 8 bit precision, both the weights and activations must be quantized. Activations are quantized with the use of a calibration set to estimate the dynamic range of the activations. Our network architecture consists of TDNN layer followed by ReLU and Batchnorm layers as mentioned in Chapter 3. In our experiments, we quantize only the weights and input activations of the TDNN layer as depicted in Figure 4.1 and the integer arithmetic is applied only to the 1D convolution.

The translation of real-numbers computation into quantized-values computation is required in order to perform the 1D convolution for quantized weights and activations. Consider two $N \times N$ matrices of real numbers r with the entries of these matrices denoted as r_α with ($\alpha = 1$ or 2). Let the product of 2 real numbers r_1 and r_2 be denoted as $r_3 = r_1 r_2$. The matrix multiplication of the 2 matrices can be denoted as:

$$r_3^{(i,k)} = \sum_{j=1}^{N} r_1^{(i,j)} r_2^{(j,k)}. \qquad \text{for } 1 \leq i, j, k \leq N \tag{4.4}$$

Using equation 4.1 for r_α in equation 4.4, we get

$$S_3(q_3^{(i,k)} - Z_3) = \sum_{j=1}^{N} S_1(q_1^{(i,j)} - Z_1) S_2(q_2^{(j,k)} - Z_2), \tag{4.5}$$

which can be rewritten as,

$$q_3^{(i,k)} = Z_3 + M \sum_{j=1}^{N} \left(q_1^{(i,j)} - Z_1\right) \left(q_2^{(j,k)} - Z_2\right), \tag{4.6}$$

where the multiplier M is defined as

$$M = \frac{S_1 S_2}{S_3}. \tag{4.7}$$

Simplifying equation 4.6, we get:

$$q_3^{(i,k)} = Z_3 + M \left(N Z_1 Z_2 - Z_1 a_2^{(k)} - Z_2 a_1^{(i)} + \sum_{j=1}^{N} q_1^{(i,j)} q_2^{(j,k)} \right), \tag{4.8}$$

where $a_2^{(k)} = \sum_{j=1}^{N} q_2^{(j,k)}$ and $a_1^{(k)} = \sum_{j=1}^{N} q_1^{(i,j)}$.

Equation 4.8 is used to multiply the quantized weights and activations in 1D convolution. The computation of $a_2^{(k)}$ and $a_1^{(i)}$ collectively takes $2N^2$ additions and the last term in equation 4.8 takes $2N^3$ operations.

As in our experiments we only perform integer arithmetic of weights and activations, equation 4.8 becomes

$$r_3^{(i,k)} = M \left(N Z_1 Z_2 - Z_1 a_2^{(k)} - Z_2 a_1^{(i)} + \sum_{j=1}^{N} q_1^{(i,j)} q_2^{(j,k)} \right), \tag{4.9}$$

with $M = S_1 S_2$. The above equation gives the floating-point value which is then added with the floating-point bias vector. The code relating to this operation is provided in

Appendix A.

Floating point operations are used in ReLU and Batchnorm layers in order to simplify the implementation, as the main focus of this paper is to only study the impact of quantization on AM weights and activations. The conventional word-recognition lattices are then generated by a Kaldi decoder (i.e. performance in Kaldi) with the use of Pytorch generated likelihoods.

4.5 Post quantization fine-tuning

Quantization is a process that reduces the precision of the model. This implies that noise is added when weights are quantized. In order to reduce the level of noise, a process of fine tuning is carried out. In this experiment, the quantized weights are first de-quantized and saved. This model is then loaded back to Kaldi and further trained for 2 epochs with a low learning rate. The process of quantizing and fine tuning is carried out in three iterations with an assumption that the final model when quantized converges to the baseline TDNN model.

5 Experiments

5.1 Experimental Setup

This section briefly describes the data used throughout all the experiments performed in this thesis, the acoustic and language models used in KWS task and different parameter reduction techniques applied for ASR task. This is followed by the overview of the evaluation metric used in ASR and KWS task.

5.1.1 Data

Training Data

The research performed for all the parameter reduction techniques for ASR tasks employs the Librispeech dataset [36]. Librispeech is a corpus of approximately 1000 hours of 16 kHz read English speech from the LibriVox project. The LibriVox project is responsible for the creation of approximately 8000 public domain audio books, the majority of which are in English. Most of the recordings are based on texts from Project Gutenberg2, also in the public domain.

The results presented for the KWS task employs models trained with 2 datasets. They are:

- Models trained with only Librispeech dataset consisting of 960 hours.
- Model trained with 150 hours of a combined dataset of AMI, Librispeech and TEDLIUM (ILT).

The AMI [37] Meeting Corpus consists of 100 hours of meeting recordings. The recordings use a range of signals synchronized to a common timeline. These include close-

Table 5.1 – The keywords present in the Logi-test set.

Keywords
Hey Alexa
Hey Blue Genie
Hey Logi
Hey Logitech
Hey Siri
Mute
OK Google
Pause
Play
Rewind Ten Seconds
Set Volume Five
Stop
Volume Down
Volume Up

talking and far-field microphones, individual and room-view video cameras, and output from a slide projector and an electronic whiteboard. The meetings were recorded in English using three different rooms with different acoustic properties, and include mostly non-native speakers.

The TED-LIUM [38] corpus is English-language TED talks with transcriptions sampled at 16kHz. It contains about 118 hours of speech.

Evaluation set

The speech recognition performance for all ASR related experiments is measured on the Librispeech test-clean evaluation set. The KWS evaluations presented are evaluated on two different test sets. The 2 test sets are (i) Librispeech test-clean evaluation set, and (ii) a subset of proprietary Logitech voice control dataset (Logi test set). The Librispeech test set consists of 40 speakers with 5.4h of data and the Logi test set consists of 12 US English native speakers and 14 keywords [39] listed in Table 5.1, an additional label for "Unknown Word" which is not part of the keywords and a label "Silence" when no speech is detected [40]. Although the test set consists of 14 commands, the results presented are evaluated on 7 keywords (DOWN, FIVE, PLAY, PAUSE, STOP, SECONDS, VOLUME), a subset of the 14 commands. These 7 keywords are selected such that they occur in both the Logi and Libri test set.

5.1.2 KWS Task

As mentioned in Chapter 1, the aim of this project is to study efficient ways to reduce the parameters of KWS/ASR engines in order to allow them to operate on the embedded devices. As hitherto mentioned, the traditional lattice based KWS exploits LVCSR decoder which requires large (in case of Kaldi - a FST) graphs to be loaded to the memory, while used during decoding. These graphs cannot be easily loaded on the embedded device due to their large size (i.e. in case of LVCSR task considering hundreds of thousands words in vocabulary, the composed HCLG graphs are of size of several hundreds of MB, which is not practical due to memory constraints of embedded devices). In addition, the lattice decoder is usually a CPU-heavy algorithm which requires considerable experience in installation, processing and debugging.

Traditional KWS approach (often denoted as query term detection) built on LVCSR decoder considers word-recognition lattices generated by the decoder and the predefined keywords are searched in the lattices. Posterior probability can then be assigned to each keyword detected in the lattice (i.e. a confidence score). This approach is considered as an offline solution since it requires generating ASR lattices which are then indexed for KWS task. This approach was not used in the evaluation experiments presented below.

A naive way to deal with the KWS detection problem is to build a G.fst graph (related to LM) using a loop of words considered in keyword list, while adding an ad-hoc penalty for recognizing a new word (e.g. a threshold between sil/sp states when passing through word arcs). In this case the G.fst is relatively small (a set of unigram probabilities) and is composed with the other H, C, L FST graphs [1]. Unfortunately the drawback of this approach is that there is not a clear definition of confidence of detected keyword (as the composed FST graph is used to generate 1-best hypothesis and alternative hypotheses (e.g. generated by a background model) are not considered. The second drawback is the use of still heavy decoder (FST based).

Therefore, we considered the third approach relying on an acoustic KWS approach. The idea behind this work is motivated by the fact that flat-start LF-MMI models (considering a monophone states as the output of the network) trained in Kaldi provide "almost" as good performance as the models trained in a standard (triphone) setup. Hossein et al. in [41] showed that using a monophone model does not impact the recognition performance of the system. As acoustic KWS approach usually works on sequence of monophones (i.e. it has been shown that context-dependent graphs do not necessarily improve the performance), we build the acoustic based KWS on

[1] http://kaldi-asr.org/doc/graph.html

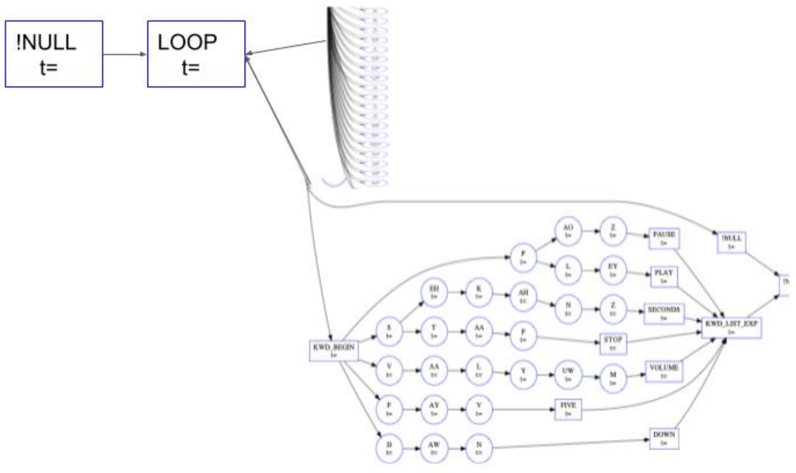

Figure 5.1 – Example graph built by acoustic KWS approach when evaluated on logi-test set with 7 keywords.

monophones as well (as explained in Section 5.1.3). In addition to monophone based outputs, low-rank matrix factorization is used in training of AM.

Acoustic models

In acoustic KWS detection system, a light decoder can be used (Viterbi based on token passing), considering a simple (HTK based) graph which consists of two hypotheses: (1) considering a keyword(s) occurring in an input acoustic sequence, (2) alternative hypothesis built using a universal background model, assuming the keyword does not occur in the input speech sequence. Finally two likelihoods are estimated and a likelihood ratio is computed which represents the confidence of the given word.

Figure 2.2 provides a high-level description of the acoustic KWS approach: filler models, background model, and model(s) related to keywords to be detected. Figure 5.1 is an example of the graph being built for our logi-test evaluation dataset considering 7 keywords.

The approach also allows the introduction of word insertion penalty (WIP) which can be used to tune the false-alarm rate of the acoustic-KWS system. Both acoustic KWS and lattice based KWS employ TDNN LF-MMI acoustic model - meaning the pseudo likelihoods estimated by TDNN model (softmax monophone outputs) are used as

Table 5.2 – Overview of AMs used in the evaluation of the Librispeech test-clean set. The number of layers in these models are 17 with the architecture same as in Table 5.5. The stride of layers 7 to 17 are same as layer 6 of the model in Table 5.5. The number of outputs of monophone models are 41 and 5984 for triphone model. All these models are used in the lattice-based KWS approach.

Model	Dataset	Training data (hours)	#Params (M)
monophone-libri	Librispeech	960	7.2
triphone-libri	Librispeech	960	20.8
monophone-ilt	AMI+Librispeech+TED (ILT)	150	7.2

emission probabilities in the decoder.

The acoustic models used in acoustic KWS and lattice-based KWS are based on TDNN-F architecture as shown in Table 5.5. The layers 7 to 17 are same as layer 6 of the TDNN-F architecture in Table 5.5. The results presented on the libri test set are evaluated only in lattice-based (naive) approach. An overview of the AMs used, training data specifications are mentioned in Table 5.2. The The AMs are trained using conventional high-resolution MFCC features with 40 coefficients with speed perturbation for data augmentation.

The KWS performance on the logi-test set is obtained for both naive lattice based KWS and acoustic KWS approaches and compared with a CNN model trained in Pytorch. The model CNN_10K is a system trained with ResNet architecture [42] and considers 10'000 keyword classification task (i.e. the network is trained to classify the whole words on training data). An overview of the AMs used for evaluating logi test set is shown in Table 5.3.

Language model

For lattice-based KWS systems, a LM is trained with the keywords to be detected along with "SIL" phone for all other keywords. Practically it considers uni-gram probabilities only when the LM is built. The acoustic based KWS does not require a LM.

5.1.3 ASR task

Acoustic model

The AMs are trained with 960h of Librispeech [36] data. The trained AMs use conventional high-resolution mef-frequency cepstral coefficients (MFCC) features i.e., 40 MFCCs are computed as each time step. In this deep learning period where large

Table 5.3 – Overview of AM models evaluated on logi-test set. In acoustic-based KWS, SLratio decoder is deployed. Different word insertion penalties (WIP) are used to penalize the detected keywords. In lattice-based KWS, Kaldi decoder is applied. The AMs in lattice-based KWS are the same as the models in Table 5.2. The performance of AMs used in both acoustic and lattice based setups is presented in Table 5.6. The CNN_10K model is trained in Pytorch.

Model	KWS Approach	Train data	Other specifics
htk	Acoustic	Librispeech (960h)	17 layers, WIP=0
htk7M+p-5	Acoustic	Libri (960h)	17 layers, WIP=-5
htk_p-5	Acoustic	Libri (960h)	10 layers, WIP=-5
monophone-libri	Lattice	Librispeech (960h)	17 layers
triphone-libri	Lattice	Librispeech (960h)	17 layers
monophone-ilt	Lattice	AMI+Libri+TED (150h)	17 layers
triphone-ilt	Lattice	AMI+Libri+TED (150h)	8 layers (dim=1024)
CNN_10K	Pytorch	Libri(960h)	10K outputs

networks are used to train models require large amounts of data. One way to increase the quantity of the training data is through data augmentation. In our experiments, audio speed perturbation [43] is applied to the acoustic data. Speed perturbation of an audio signal produces warped time signal and changes the duration of the signal. When the speed of the signal is reduced (less than 1x), the energy of the signal shifts towards the lower frequencies. This results in very small energies in the higher Mel bins of the MFCCs. I-vectors are not used in these experiments.

A detailed description of the TDNN architecture with 7 layers is presented in Table 5.4 and TDNN-F architecture with 7 layers is presented in Table 5.5. The hidden dimension for a TDNN/TDNN-F model is 625. Models with 10 layers and 17 layers have the same respected TDNN/TDNN-F context/stride. A TDNN layer is followed by a ReLU [14; 44] and a Batchnorm layer [45].

All the experiments conducted to reduce parameters of TDNN-based acoustic models are trained with Kaldi toolkit (i.e. nnet3 architecture). AMs are trained with the LF-MMI training framework, considered to produce state-of-the-art performance for hybrid ASR systems. In the study, we do not only consider conventional triphone systems but also a monophone based ASR system. In the former case, the output layer consists of senones obtained from clustering of context-dependent phones. In the latter case, the output layer consists of only monophone outputs, which can be considered as yet another approach to reduce the computational complexity of ASR systems. The triphone-based AM uses position-dependent phones which produces a total of 346 phones including the silence and noise phones. The monophone-based

Table 5.4 – TDNN architecture description with 7 layers. Xent: Cross entropy layer used to avoid over-fitting. The output dimension of all the layers is 625 and the output dimension of the "Output" layer is 41 for monophone based model and 5984 for triphone based model.

Layer	**Layer context**	**Subsampling factor**
TDNN-1	[0]	1
TDNN-2	{-1,1}	1
TDNN-3	{-1,1}	3
TDNN-4	{-3,3}	1
TDNN-5	{-3,3}	1
TDNN-6	{-3,3}	1
TDNN-7	{-3,3}	1
Prefinal chain	[0]	1
Prefinal-xent chain	[0]	1
Output	[0]	1

Table 5.5 – TDNN-F architecture description with 7 layers. Xent: Cross entropy layer used to avoid over-fitting. The output dimension of all the layers is 625 and the output dimension of the "Output" layer is 41 for monophone based model and 5984 for triphone based model.

Layer	**Time stride**
TDNN-1	0
TDNNF-2	1
TDNNF-3	1
TDNNF-4	1
TDNNF-5	0
TDNNF-6	3
TDNNF-7	3
Prefinal chain	[0]
Prefinal-xent chain	[0]
Output	-

AM uses position-independent phones which comprises 41 phones. The output of the triphone-based AM produces 5984 states while the monophone-based AM produces 41 states.

The quantization experiments are performed in Pytorch. Quantization experiments are carried out for 16 bit and 8 bit integers in symmetric mode. As discussed in Chapter 3, the model and the features from Kaldi are loaded as Pytorch tensors with the help of the C++ wrapper.

Language Model

The LMs trained on Librispeech are available to be downloaded from Open Speech and Language Resources (Open SLR)[2]. It is a website that hosts speech and language resources like software and training data related to speech recognition. The LM trained on Librispeech uses 14500 public domain books with a 200K unique words vocabulary. The website provides 3-gram and 4-gram models to download. In addition to it, the 3-gram models are pruned at different thresholds such as 10^{-7} and 3×10^{-7}. In the monophone setup, the TDNN and TDNN-F models use the small (3×10^{-7}) 3-gram pruned model for decoding and the 4-gram model for lattice-rescoring while decoding in Kaldi. The experiments performed to compare varying layers of TDNN and TDNN-F AM employ a large LM. The quantization experiments performed in both monophone and triphone setup employ the small LM.

5.1.4 Evaluation metric

The performance of an ASR system is presented as Word Error Rate (WER). It is based on the Levenshtein distance at the word level. It can be viewed as a string matching problem where two sequences of symbols are matched through dynamic programming. The symbols in this case are the words of a language. WER finds the distance between the word sequence recognized by the ASR and the reference word sequence using dynamic string alignment i.e., it finds the number of edits (substitutions, deletions, insertions) required to go from the recognized word sequence to the reference word sequence. Given the recognized and reference sequences of words, WER is computed as:

$$WER = \frac{S + D + I}{N}, \tag{5.1}$$

[2]http://www.openslr.org/11/

Table 5.6 – Comparison of TDNN-F model with varying number of layers in the monophone setup when a large LM was used for rescoring.

Model	No. of layers	Params (M)	WER %
TDNN	7	7.9	5.4
TDNN-F	7	3.14	7.3
TDNN-F	10	4.35	5.7
TDNN-F	17	7.1	5.2

where S is the number of words that are substituted, D is the number of deletions, I is the number of insertions, $N = S + D + C$ is the total number of words in the reference and C is the number of correct words. A lower WER implies better accuracy for the ASR system.

As mentioned earlier, KWS is a detection task and its performance is usually presented in the form of Receiver Operating Characteristic (ROC) curve and Figure-of-Merit (FOM) metric proposed by the National Institute of Standards and Technology (NIST) and described in the HTK book [46]. A ROC curve is a plot of true positive (TP) rate against false positive (FP) or also called as false alarm (FA) at various thresholds. A TP means that the model correctly predicts the keyword and an FP implies that the model fails (misses) to predict the true keyword. The ROC curve shows the average (over keywords) performance (TP) for a given false-alarm rate. The number of true detections above these thresholds is defined as the detection probability estimate. The results show the TP over a range of 0 to 10 FA per hour. Hence for a given false alarm rate, the system that has a higher TP value is better. This allows for a more stable statistic of performance and will be used as a figure of merit for comparing different systems.

In addition to the ROC curve, FOM for each keyword is also plotted. The FOM is an upper-bound estimate on keyword spotting accuracy averaged over 1 to 10 false alarms per hour. This plot helps to better understand the performance of each keyword for a given model. While ROC shows the average performance of a model over all keywords given a false alarm rate, the FOM shows the performance of a model for each keyword.

5.2 TDNN-F experiments

The experiments for comparison of the number of parameters and its impact on the performance of the ASR while using TDNN and TDNN-F is presented in this section.

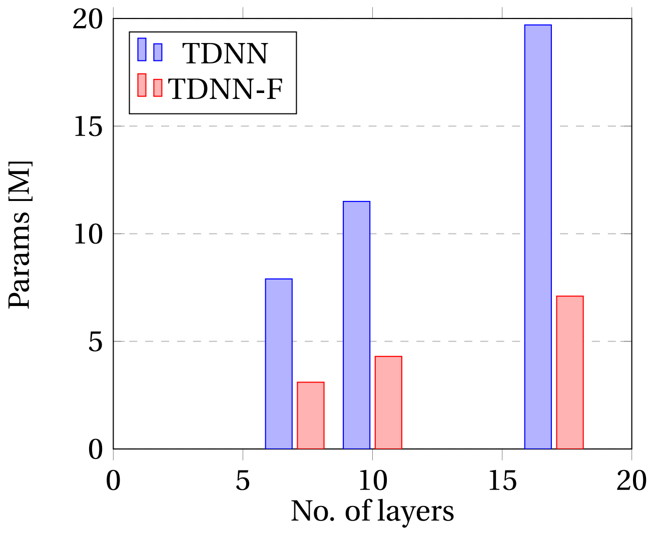

Figure 5.2 – Comparison of # parameters in TDNN and TDNN-F with varying number of layers in the monophone setup when a large LM was used for rescoring.

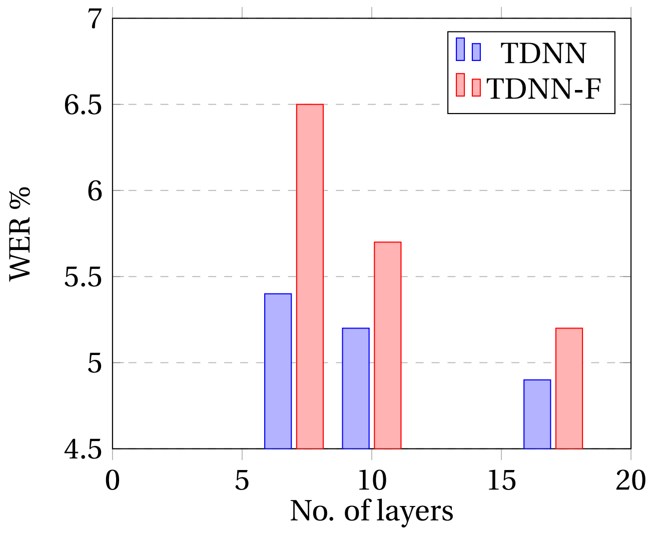

Figure 5.3 – Comparison of Word Error Rate (WER %) in TDNN and TDNN-F with varying number of layers in the monophone setup when a large LM was used for rescoring.

Table 5.7 – Comparing parameter reduction techniques for monophone-based TDNN acoustic model: Quantization (bits), Weight Quantization (WQ), Activation Quantization (AQ), Number of parameters, model size and Word-Error Rate (WER) [in %] when a small LM was used for decoding.

Model	Quantization (bits)	WQ	AQ	Params (M)	Size	WER %
Baseline TDNN	-	No	No	7.9	1x	8.1
TDNN	16	Yes	No	7.9	0.5x	10.7
TDNN	8	Yes	No	7.9	0.25x	10.7
TDNN	8	Yes	Yes	7.9	0.25x	17.3
TDNN - fine tuned	8	Yes	Yes	7.9	0.25x	18.5
TDNN-F (7 layers)	-	No	No	3.14	0.4x	9.8

In this study, TDNN and TDNN-F models were trained by increasing the number of layers from 7 up to 17 layers in the monophone setup. Table 5.6 shows that by using twice as many layers as TDNN in TDNN-F, the same number of parameters (7M) in baseline TDNN model is reached with an improved performance. The ASR performance presented in this table is obtained from rescoring the ASR outputs with a large LM trained on Librispeech. Figure 5.2 shows the comparison of the number of parameters in TDNN and TDNN-F when the number of layers in the model are increased up to 17. For TDNN models, doubling the number of layers increases the total number of parameters three times, whereas in case of TDNN-F models, doubling the number of layers doubles the number of parameters (while the performance is still close to the baseline TDNN model). Figure 5.3 shows the comparison of the WER for TDNN and TDNN-F models when the number of layers in the model are increased up to 17. Using few layers in both TDNN and TDNN-F shows that the performance of the TDNN-F is degraded by 1% (absolute in WER) but using more layers in TDNN and TDNN-F shows that the performance of TDNN-F model is close to the TDNN model.

5.3 Parameter reduction experiments

We compare floating-point vs. integer arithmetic inference for the TDNN model with different quantization types (16-bit and 8-bit integer) and different quantization schemes, as discussed in Section 3.2. We also compare the quantization technique with the low-rank matrix factorization technique used during the training of the model.

Table 5.7 shows that weight-only quantization reduces the model size by 50% without a significant impact on the performance of the monophone-based AM. Quantizing

Table 5.8 – Comparing parameter reduction techniques for triphone-based TDNN acoustic model: Quantization (bits), Weight Quantization (WQ), Activation Quantization (AQ), Number of parameters, model size and Word Error Rate (WER) [in %] when a small LM was used for decoding.

Model	Quantization (bits)	WQ	AQ	Params (M)	Size	WER %
Baseline TDNN	-	No	No	15.4M	1x	6.3
TDNN	16	Yes	No	15.4M	0.5x	10.0
TDNN	8	Yes	No	15.4M	0.25x	11.4
TDNN	8	Yes	Yes	15.4M	0.25x	11.2
TDNN - fine tuned	8	Yes	Yes	15.4M	0.25x	11.3
TDNN-F	-	No	No	6.2M	0.4x	6.9

Table 5.9 – Comparison of quantization error for the monophone and triphone-based TDNN acoustic models presented in Table 5.7 and Table 5.8.

Quantization (bits)	Monophone	Triphone
16	12.9	2.5
8	13.5	22.7

both weights and activations reduces the model size with an increased WER compared to weight only quantization. Table 5.8 shows that quantizing both weights and activations outperforms the weight-only quantization in the triphone system. In both monophone and triphone systems, post quantization fine-tuning does not show any impact. The TDNN-F model reduces the model size by 40% with a drop in the recognition performance of 2.7% (in absolute WER) compared to the baseline TDNN. However, compared to the 8-bit and 16-bit quantized model, the WER degradation of TDNN-F is negligible (10.7% WER for the quantized model vs 10.8% WER of TDNN-F).

5.4 Quantization error

The quantization error is measured as the norm between the original weights and its de-quantized version

$$\text{Quantization error} = \sum_{i,j} \left(w_{ij} - \tilde{w}_{ij}\right)^2, \tag{5.2}$$

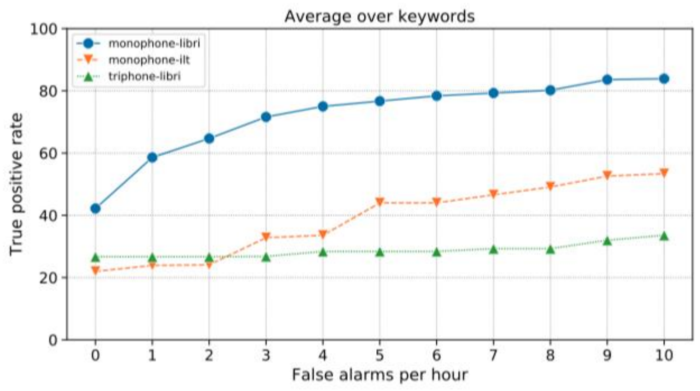

Figure 5.4 – ROC curve evaluated on Librispeech test-clean set with lattice-based KWS with 7 keywords. The AMs used are described in Table 5.2.

which is the Frobenius norm between the original weight matrix and the reconstructed weight matrix. The row and column indices are referred with the variables i and j. The error measures the shift in parameters due to the quantization process.

Table 5.9 shows the error for monophone and triphone-based AMs with respect to int8 and int16 quantization. As expected, the 16-bit quantization provides a smaller error rate compared to 8-bit quantization. However, the difference in the norms between 8-bit and 16-bit quantization is much higher for triphone systems. This high variation of the error in the triphone system is due to its large number of outputs.

5.5 KWS experiments

As mentioned above, reduction techniques are also presented for KWS task. The results presented below for different evaluation sets are evaluated with 7 keywords (DOWN, FIVE, PLAY, PAUSE, STOP, SECONDS, VOLUME). The below plots are evaluated on 2 test sets as described in Section 5.1.1. The AMs used in the evaluation of the Librispeech test-clean set are described in Table 5.2. The AMs used in the evaluation of the logi-test set are described in Table 5.3.

Figure 5.4 shows the ROC curve for three different models generated for 7 keywords evaluated on the Librispeech test-clean dataset with lattice-based KWS. As the plot indicates: using a model trained Librispeech offers a better performance as the test data matches the train data (i.e. similar acoustic conditions). It also shows that in this task, monophone setup yields better performance than triphone based setup.

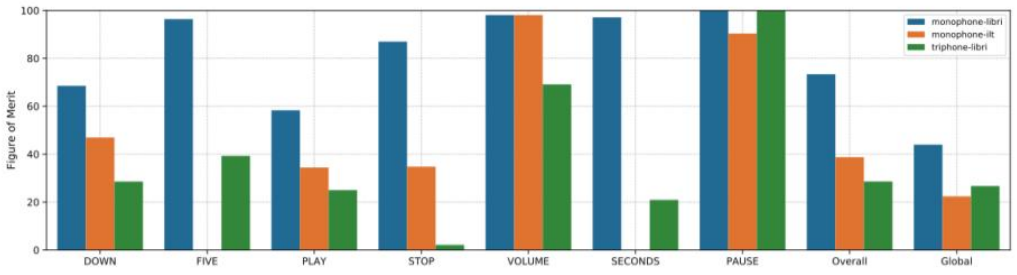

Figure 5.5 – FOM plot for 7 keywords evaluated on Librispeech test-clean set with lattice-based KWS. The AMs used are described in Table 5.2.

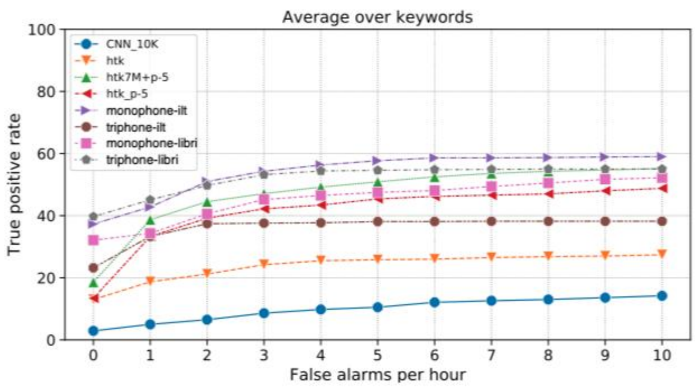

Figure 5.6 – ROC curve evaluated on Logitech test set with 7 keywords. The models in the curve are described in Section 5.1.2 Table 5.3.

Figure 5.5 shows the FOM plot for each of the seven keywords evaluated on the Librispeech test-clean set on lattice-based KWS as mentioned in Section 5.1.2. As mentioned in Section 5.1.4, the plot shows the average merit for each keyword. This helps us understand how a model performs on a keyword. As shown in the plot, the Librispeech model with monophone outputs has the highest merit for each keyword with an overall merit of around 72 points and the performance of the Librispeech model in triphone setup has performance similar to its monophone counterpart for certain keywords. This could be due to the variation in the pronunciations in these two setups. In this plot we see that some keywords such as VOLUME, PAUSE have a very high merit (close to 100) for all the models which means that the keyword appears many times in the test set and the model doesn't fail to predict. There are also some keywords such as FIVE, SECONDS that have a very low merit (close to 0) for model

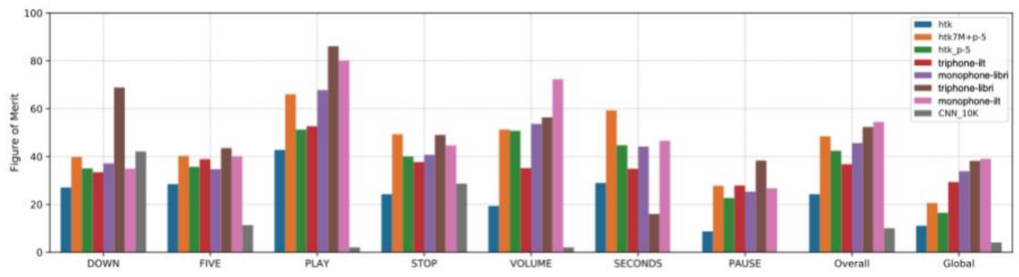

Figure 5.7 – FOM plot for 7 keywords evaluated on Logitech test set. The models in the curve are described in Section 5.1.2 Table 5.3.

trained with the combined dataset. After an investigation, we see that those words do not appear quite often in the test and the model has not seen many examples or has failed to predict. Since the FOM plot shows for which keywords which model performs better, the performance of the model for keywords that do not perform well can be improved by adding more pronunciation variants into the dictionary.

Figure 5.6 shows the ROC curve for the aforementioned models (described in Section 5.1.2 Table 5.3) evaluated on the Logitech test set. The curve shows models evaluated in lattice-based, acoustic based KWS and the model trained in Pytorch. As the plot shows, using a model trained with a combination of AMI, Librispeech and TEDLIUM (ILT) has a slightly better performance of 2 points than using a model trained only on the Librispeech dataset as the test data matches better the train data. The plot also shows that although the acoustic KWS (htk7M+p-5 model) has a good performance, the lattice-based KWS (monophone-ilt model) is the one with the best performance. It also shows that the Librispeech model with triphone outputs is closer to the ILT model since there are many outputs in the triphone outputs which implies that there many variations of the pronunciations in this setup.

Figure 5.7 shows the FOM plot for each of the seven keywords evaluated on the Logitech test set. The description of the models in the plot are provided in Table 5.3. Although the combined dataset (AMI+Libri+TED (ILT)) model with monophone outputs has an Overall merit of around 55, the Librispeech model with triphone outputs has a high merit for majority of the keywords. This shows that using a triphone setup gives a better performance and the performance of the ILT model can be improved by using triphone outputs. The plot shows the performance of the acoustic KWS is close to the lattice KWS and can be improved by tuning the WIP.

6 Conclusions

This study presented the effect of using different parameter reduction techniques of acoustic models on ASR and KWS tasks. The ASR experimental results reveal that the parameter-quantization can reduce the model size significantly while preserving a reasonable word recognition performance. TDNN-F models provide a better performance when the number of layers is higher than the TDNN models. The experiments also reveal that the number of parameters of a TDNN-F model with twice as many layers as TDNN remains the same. As DNN requires many layers to model complex relationships, using TDNN-F does not have a huge impact on the footprint of the model. While using monophone based outputs does not have a huge impact on the recognition accuracy for ASR, KWS experiments reveal that using triphone based outputs has a better FOM depending on the test case. The KWS experiments also show that a model trained with a combination of multiple dataset has a relatively better overall performance.

6.1 Future Work

Effect of quantization on KWS task has to be studied as a small footprint is required on the embedded device. Quantization of the acoustic models can be further explored through fusing the TDNN, ReLu and Batchnorm layers. Since fine-tuning did not bring any significant improvements in our experiments, our future work will consider an implementation of the quantization-aware training. The quantization experiments are conducted in Pytorch, while the acoustic models are developed using the popular Kaldi toolkit. Implemented C++ wrappers allowing to interface parameters of the Kaldi-based DNN acoustic models in Pytorch will be offered to other researchers through a Github project.

A Psuedo code for Quantization

A.1 Calculation of scale and zero-point

The scale and zero-point of a tensor are calculated according to the formula in Section 3.1. The code to calculate the scale and zero-point for 8 bit quantization in asymmetric mode is shown below. The below function requires the minimum value, maximum value of the tensor (weight matrix or feature vector).

```
import torch
import numpy as np


def calcScaleZeroPointInt8(min_val, max_val):
  # Calc Scale and zero point of next
    qmin = -127
    qmax = 127

    scale_next = (max_val - min_val) / (qmax - qmin)

    initial_zero_point = qmin - min_val / scale_next
    zero_point_next = 0
    if initial_zero_point < qmin:
        zero_point_next = qmin
    elif initial_zero_point > qmax:
        zero_point_next = qmax
    else:
        zero_point_next = initial_zero_point

    zero_point_next = int(zero_point_next)
    return scale_next, zero_point_next
```

A.2 Quantization of a tensor

Once the scale and zero-point are calculated, the tensor is quantized according to the equation described in Section 3.1. The below code shows the quantization of a tensor to 8-bit. The function takes the input tensor, its scale and zero-point and returns the quantized tensor, scale and zero-point. Returning of the scale and zero-point will be useful during de-quantization.

```
def quantize_tensor_int8(tensor, scale, zp):
    t = tensor/scale + zp
    t = t.round()
    t = t.to(torch.int8)
    return (t, scale, zp)
```

A.3 Forward pass for quantized tensors

Once the input and the weight matrix are quantized to 8 bits, the forward pass requires the Integer-arithmetic-only matrix multiplication. The code for the forward pass is given below.

```
def forward(input, input_scale=None, input_zpt=None):
    scale, zero_point =
      calcScaleZeroPointInt8(self.linear_params_.data.min(),
      self.linear_params_.data.max())

    qweight_int8 = quantize_tensor_int8(self.linear_params_, scale,
      zero_point)
    dequant_weight = dequantize_tensor(qweight_int8)
    print("Norm of weights and dequantized weights:
      {}".format(torch.norm(self.linear_params_.data - dequant_weight)))

    padded_input_uint8 = quantize_tensor_int8(padded_input, input_scale,
      input_zpt)
    qweight_int32 = qweight_int8[0].to(torch.int32)
    padded_input_int32 = padded_input_uint8[0].to(torch.int32)
    x1 = padded_input_int32.matmul(qweight_int32.t())
    x2 = input_zpt*qweight_int32.sum(1)
    x3 = zero_point*padded_input_int32.sum(2)
    N = self.linear_params_.shape[1]
    x = (x1-x2).squeeze(0).t() - x3.reshape(-1)
    x = x + N*zero_point*input_zpt
    x = x.t().unsqueeze(0).float()*(input_scale*scale)
    x = x+self.bias_.reshape(1, -1)

    return x
```

A.4 Conversion of integer value to floating-point value

De-quantization is performed when weight only quantization is used so that the inference is carried out in floating point precision. The code to convert the quantized value back to floating value is given below.

```
def dequantize_tensor(quant_tensor):
    q_tensor = quant_tensor[0]
    scale = quant_tensor[1]
    zero_point = quant_tensor[2]
    dequant_tensor = (q_tensor.float() - zero_point) * scale

    return dequant_tensor
```

Bibliography

[1] Y. LeCun, Y. Bengio, *et al.*, “Convolutional networks for images, speech, and time series,” *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[2] A. Graves, A.-r. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” in *2013 IEEE international conference on acoustics, speech and signal processing*, pp. 6645–6649, IEEE, 2013.

[3] V. Peddinti, D. Povey, and S. Khudanpur, “A time delay neural network architecture for efficient modeling of long temporal contexts,” in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[4] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, *et al.*, “The kaldi speech recognition toolkit,” in *IEEE 2011 workshop on automatic speech recognition and understanding*, no. CONF, IEEE Signal Processing Society, 2011.

[5] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.

[6] D. Povey, G. Cheng, Y. Wang, K. Li, H. Xu, M. Yarmohammadi, and S. Khudanpur, “Semi-orthogonal low-rank matrix factorization for deep neural networks.,” in *Interspeech*, pp. 3743–3747, 2018.

[7] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems*, pp. 8024–8035, 2019.

[8] P. Motlicek, F. Valente, and I. Szoke, “Improving acoustic based keyword spotting

using lvcsr lattices," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4413–4416, IEEE, 2012.

[9] L. Burget, O. Glembek, P. Schwarz, and M. Karafiat, "Hmmtoolkit stk," 2006.

[10] A. Prasad, P. Motlicek, and S. Madikeri, "Quantization of acoustic model parameters in automatic speech recognition framework," 2020.

[11] M. Mohri, F. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech & Language*, vol. 16, no. 1, pp. 69–88, 2002.

[12] M. Mohri, F. Pereira, and M. Riley, "Speech recognition with weighted finite-state transducers," in *Springer Handbook of Speech Processing*, pp. 559–584, Springer, 2008.

[13] M. Riley, C. Allauzen, and M. Jansche, "Openfst: An open-source, weighted finite-state transducer library and its applications to speech and language," in *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Tutorial Abstracts*, pp. 9–10, 2009.

[14] D. Povey, V. Peddinti, D. Galvez, P. Ghahremani, V. Manohar, X. Na, Y. Wang, and S. Khudanpur, "Purely sequence-trained neural networks for asr based on lattice-free mmi.," in *Interspeech*, pp. 2751–2755, 2016.

[15] A. Graves, N. Jaitly, and A.-r. Mohamed, "Hybrid speech recognition with deep bidirectional lstm," in *2013 IEEE workshop on automatic speech recognition and understanding*, pp. 273–278, IEEE, 2013.

[16] H. Sak, A. W. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," 2014.

[17] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Transactions on audio, speech, and language processing*, vol. 22, no. 10, pp. 1533–1545, 2014.

[18] K. Veselỳ, A. Ghoshal, L. Burget, and D. Povey, "Sequence-discriminative training of deep neural networks.," in *Interspeech*, vol. 2013, pp. 2345–2349, 2013.

[19] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[20] D. Jurafsky, *Speech & language processing.* Pearson Education India, 2000.

[21] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur, "Recurrent neural network based language model," in *Eleventh annual conference of the international speech communication association*, 2010.

[22] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning.* MIT press, 2016.

[23] D. Can and M. Saraclar, "Lattice indexing for spoken term detection," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 8, pp. 2338–2347, 2011.

[24] I. Szöke, P. Schwarz, P. Matějka, L. Burget, M. Karafiát, and J. Černockỳ, "Phoneme based acoustics keyword spotting in informal continuous speech," in *International Conference on Text, Speech and Dialogue*, pp. 302–309, Springer, 2005.

[25] D. R. Miller, M. Kleber, C.-L. Kao, O. Kimball, T. Colthurst, S. A. Lowe, R. M. Schwartz, and H. Gish, "Rapid and accurate spoken term detection," in *Eighth Annual Conference of the international speech communication association*, 2007.

[26] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[27] J. H. Wong and M. Gales, "Sequence student-teacher training of deep neural networks," in *Proc. of Interspeech 2016*, 2016.

[28] F. Keith, W. Hartmann, M.-H. Siu, J. Ma, and O. Kimball, "Optimizing multilingual knowledge transfer for time-delay neural networks with low-rank factorization," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4924–4928, IEEE, 2018.

[29] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv preprint arXiv:1806.08342*, 2018.

[30] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 535–541, 2006.

[31] J. Xue, J. Li, and Y. Gong, "Restructuring of deep neural network acoustic models with singular value decomposition.," in *Interspeech*, pp. 2365–2369, 2013.

[32] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE transactions on acoustics, speech, and signal processing*, vol. 37, no. 3, pp. 328–339, 1989.

[33] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

[34] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283, 2016.

[35] D. Can, V. R. Martinez, P. Papadopoulos, and S. S. Narayanan, "Pykaldi: A python wrapper for kaldi," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5889–5893, IEEE, 2018.

[36] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5206–5210, IEEE, 2015.

[37] J. Carletta, S. Ashby, S. Bourban, M. Flynn, M. Guillemot, T. Hain, J. Kadlec, V. Karaiskos, W. Kraaij, M. Kronenthal, *et al.*, "The ami meeting corpus: A pre-announcement," in *International workshop on machine learning for multimodal interaction*, pp. 28–39, Springer, 2005.

[38] A. Rousseau, P. Deléglise, and Y. Esteve, "Ted-lium: an automatic speech recognition dedicated corpus.," in *LREC*, pp. 125–129, 2012.

[39] N. Sacchi, A. Nanchen, M. Jaggi, and M. Cernak, "Open-vocabulary keyword spotting with audio and text embeddings," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, no. CONF, 2019.

[40] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint arXiv:1804.03209*, 2018.

[41] H. Hadian, H. Sameti, D. Povey, and S. Khudanpur, "End-to-end speech recognition using lattice-free mmi.," in *Interspeech*, pp. 12–16, 2018.

[42] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[43] T. Ko, V. Peddinti, D. Povey, and S. Khudanpur, "Audio augmentation for speech recognition," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[44] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[45] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[46] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. Odell, D. Ollason, D. Povey, *et al.*, "The htk book," *Cambridge university engineering department*, vol. 3, p. 75, 2006.

Curriculum Vitae

Name:	Amrutha Prasad
Qualification:	Bachelors in Engineering (Electronics and Communication Engineering)

Academic Background

- February 2019 - Present: Master in Artificial Intelligence from Distance University, Switzerland
- 2011-2015: Bachelor of Engineering (Electronics and Communication) from Vivekananda Institute of Technology, Bengaluru, (affiliated to Visvesvaraya Technological University, Belgaum) India with 73% of total marks (equivalent to 5.0/6 GPA in Swiss universities) over all semesters and 69% of total marks over the final 4 semesters)
- 2009-2011: Pre-University from Seshadripuram Composite Pre-University College, Bengaluru, India (Marks obtained: 78%)
- 2008-2009: Secondary School from Nirmala Rani High School, Bengaluru, India (Marks obtained: 89%)

Experience

- March 2019 - Present: Master Student at Idiap Research Institute, Martigny, Switzerland
- May 2017 - February 2019 : Software developer at recapp IT AG, Visp, Switzerland
- November 2016 - November 2017 :   Research Trainee at Idiap Research Institute,

Martigny, Switzerland with Dr. Petr Motlicek

● August - December 2014:	Trainee Engineer at BSNL (Bharat Sanchar Nigam Limited), Bengaluru, India
● February - April 2014:	Trainee Engineer at BSNL, Bengaluru, India
● August - December 2013:	Trainee Engineer at BSNL, Bengaluru, India

Publications

- Petr Motlicek, et al., "AM-FM Decomposition of Speech Signal: Applications for Speech Privacy and Diagnosis", 2019.
- Amrutha Prasad et al., "Quantization of Acoustic Model Parameters in Automatic Speech Recognition Framework", submitted to Interspeech 2020.

Achievements

- Completed Foundations of Data Science course from Edx. Received certifications for:
  - Foundations of Data Science: Computational Thinking with Python
    https://courses.edx.org/certificates/49cbeaf9d4664c88bf93cbc94e1202ef
  - Foundations of Data Science: Inferential Thinking by Resampling
    https://courses.edx.org/certificates/25b529839bae43619052810a700274ac
  - Foundations of Data Science: Prediction and Machine Learning
    https://courses.edx.org/certificates/046e642ec9d84929bf6c2625f32539b7
- Received certification from Coursera on completing:
  - Mathematics for Machine Learning: Linear Algebra by Imperial College London
    https://www.coursera.org/account/accomplishments/verify/BC55LJNVYAC6
  - Mathematics for Machine Learning: Multivariate Calculus by Imperial College

London
https://www.coursera.org/account/accomplishments/verify/VN6E85A7C475

- Passed Bachelors in Engineering with First Class Distinction (FCD) in semesters I to V and with First Class in all other semesters
- Received BSNL-AICTE certifications during the training programme at BSNL:
  - BSNL-AICTE Silver Certified Engineer (December 2013)
  - BSNL-AICTE Gold Certified Engineer (April 2014)
  - BSNL-AICTE Platinum Certified Engineer (December 2014)

Professional Skills

- Programming Languages: Python, Javascript, and HDL.
- Experiences with Libraries and Frameworks: Angular, NativeScript, Django, Jquery, Bootstrap
- Development Tools: Visual Studio Code, Vim and Sublimetext as editors and Git for version control
- Written and Spoken Languages
  - Fluent: English, Kannada, Tamil
  - Intermediate level: French (lessons regularly taken from Martigny commune)

Projects

- *Developed an android mobile application using NativeScript at recapp:*
  - Developed a front-end system which serves as an input to the ASR system
- *Development of a full-stack web application at recapp:*
  - A single page application that allows users to add and view feedback.
  - Developed a front-end system using Angular4

    - Developed a back-end system using Nodejs and MongoDB
- *Development of web server for Automatic Speech Recognition (ASR)* at Idiap:
    - Developed a Django-based web server (a python-based web framework) with a web-based front-end using HTML, Jquery and Bootstrap
    - It allows users to upload (multiple) audio files and receive the output from a speech-to-text engine
- *Language Modelling for multiple domains* at Idiap:
    - Trained a language model for ASR from multiple data sources (Common Crawl, TED talks, EuroParl, etc.) to handle different domains of speech input
    - Contributed to the ASRT toolkit (https://github.com/idiap/asrt) to normalize the data (*github username*: amrutha-p)
- *Design and Simulation of Sigma Delta Analog to Digital Converter*(ADC): As a part of our final year engineering project we designed a module in Matlab to simulate and study the efficiency of the Sigma Delta ADC

Personal Information

Date of Birth:	29-05-1993
Nationality:	Indian
Status in Switzerland:	Resident with B Permit
Residence Address:	Rue des Avouillons, 12
	CH-1920, Martigny, Switzerland
Phone (Mobile):	+41 767936507
E-mail:	amrutha.prasad29@gmail.com