Felicia Rahim

Semi-supervised learning for Automatic Speech Recognition

Master's thesis in Electronics Systems Design and Innovation
Supervisor: Giampiero Salvi
June 2020

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Felicia Rahim

Semi-supervised learning for Automatic Speech Recognition

Master's thesis in Electronics Systems Design and Innovation
Supervisor: Giampiero Salvi
June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Sammendrag

Denne masteroppgaven undersøker et talegjenkjenningssystem som trent på en delvis annotert database innenfor fagområdet talegjenkjenning (ASR). Et dypt nevralt nettverk (DNN) klassifiserte tilstander som tilhørte individuelle kontekstuavhengige fonemer (CI) og kontekstavhengige fonemer (CD). DNN-ene ble brukt i en lærer-student-metode (T/S). Lærermodellen og studentmodellen ble trent på to separate DNN. De akustiske modellene ble trent med MFCC og fMLLR informasjonsvektorer. I denne oppgaven ble nøyaktighetsraten til riktig klassifiserte fonemstilstander, fonem feilrate og forvirringsmatriser evaluert på TIMIT talekorpus. I tillegg ble lærermodellen som er trent på en manuelt annotert database evaluert mot studentmodellen som er trent på automatisk dannede annotasjoner.

Resultatene viser at de akustiske modellene oppnår høyst nøyaktighetsrate med fMLLR informasjonsvektorer. Bruk av CI fonemer gir også større nøyaktighet enn det bruk av CD fonemer gjør. Det T/S nettverket som gir høyest ytelse er trent på fMLLR informasjonsvektorer med CI fonemer, og gir en nøyaktighetsrate på 63.64% for riktig klassifiserte fonemstilstander, og en fonem feilrate på 27.47% for studentmodellen. Det nettverket som har værst ytelse var trent på MFCC informasjonsvektorer med CD fonemer, og har en nøyaktighetsrate på 35.02% for riktig klassifiserte fonemstilstander og en fonem feilrate på 39.77% for studentmodellen.

Abstract

This thesis explores semi-supervised learning for automatic speech recognition (ASR) through a teacher-student (T/S) learning technique. Frame-by-frame classifiers were implemented with deep neural networks (DNNs), using either monophones or triphones as targets. The teacher model and the student model were trained on two separate DNNs. The acoustic models were trained on Mel-frequency cepstral coefficients (MFCC) and feature-space maximum likelihood linear regression (fMLLR) features. In this work, frame-by-frame state accuracy, phoneme error rate (PER), and confusion matrices were evaluated on the TIMIT speech corpus. Additionally, the teacher model trained with hard targets was evaluated against the student model, which was trained on soft targets.

The obtained results indicate that the T/S network achieves the highest accuracy when trained on fMLLR features. Using monophones over triphones provided higher accuracy as well. The best performing T/S network trained on fMLLR features and monophone targets, and yielded a relative frame accuracy rate of 63.64% and a PER of 27.42% on the student model. Our experiment's worst-performing T/S network had a frame accuracy rate of 35.02% and a PER of 39.77% on the student model when trained with MFCCs features and triphone targets.

Preface

This Master's thesis is the terminating project of a Master's degree in Electronic systems design and innovation (ELSYS) at the Norwegian University of Science and Technology (NTNU) in Trondheim. The thesis was carried out from September 2019 to June 2020.

The Department of Electronics System provided the thesis problem in the Faculty of information technology and electrical engineering at NTNU and Professor Giampiero Salvi. The project aimed to implement semi-supervised learning for automatic speech recognition. The thesis evolved into having more of an analytic purpose rather than an experimental purpose.

I want to thank my supervisor, Professor Giampiero Salvi, for guiding and providing me with the database used in this thesis. I would also like to thank Abdolreza Sabzi Shahrebabaki for the help provided in the first months of the thesis.

Table of Contents

List of Tables

List of Figures

Abbreviations

AM	=	Acoustic model
ANN	=	Artificial neural network
ASR	=	Automatic speech recognition
CD	=	Context dependent
CI	=	Context-independent
CM	=	Confusion matrix
CMN	=	Cepstral mean normalization
CNN	=	Convolutional neural networks
DNN	=	Deep neural network
fMLLR	=	Feature-space maximum likelihood linear regression
GMM	=	Gaussian mixture model
HMM	=	Hidden markov model
KL	=	Kullback-Leibler
LSTM	=	Long short-term memory
MLP	=	Multilayer perceptron
NN	=	Neural network
PDF	=	Probability density function
PER	=	Phoneme error rate
ReLU	=	Rectified linear unit
RNN	=	Recurrent neural network
SSL	=	Semi-supervised learning
SS	=	Supervised learning
T/S	=	Teacher-student

Chapter 1

Introduction

1.1 Motivation

Automatic speech recognition (ASR) technology can be found in many applications. These include voice search on mobile and computer devices, or interaction with smart home devices such as Google Home and Amazon Echo for real-world applications such as personal assistance or shopping to name a few.

Despite the vast use of ASR, the technology has some limitations. Machine learning algorithms require hours of transcribed data of speech recordings to achieve an acceptable speech recognition accuracy. Not only is it time-consuming to produce such annotations, but it is also expensive to collect. A way of dealing with limited transcribed data is by implementing semi-supervised learning to the ASR system, which trains acoustic models on partly annotated data.

1.2 Problem description

This thesis aims to study a semi-supervised learning algorithm for ASR using a teacher-student (T/S) technique and to optimize the acoustic models of the T/S network. ASR model A is trained on a full set of manually annotated data, and is later used to produce labels for a broader set of non-annotated data. Afterward, model B is trained on a full data set (with the automatically created annotations). Additionally, for the optimization part; various features are to be evaluated against each other and evaluate context-dependent phonemes against context-independent phonemes.

1.3 Outline

Chapter 2 provides the theoretical foundation and main principles of ASR, acoustic models, and semi-supervised learning. For Chapter 3, state-of-the-art research in speech recognition is presented. Methods implemented and tested in this thesis are described in Chapter 4. Chapter 5 introduces the TIMIT speech corpus used for evaluating the acoustic models used in our work. Experiments, the actual parameters and toolkits used in methods are described in Chapter 6. Chapter 7 and 8 provides the results obtained in this work, discussions around the results and suggestions for future work. Lastly, Chapter 9 provides the conclusion.

Chapter 2

Theory

This chapter describes the fundamental theory behind semi-supervised learning for automatic speech recognition (ASR). Section 2.1 provides a description of phonetics and phonetic models. In Section 2.2, general ASR theory is presented. Section 2.3 and 2.4 gives an explanation of feature extraction and acoustic models used to recognize phonemes. Finally, in Section 2.5, semi-supervised learning is explained.

2.1 Phonetics

Phonetics is the study of speech sounds and their production, classification, and transcription. Speech sounds are based on a sequence of phonemes, where the phonemes are discrete sound segments and are linked in time.

Phonemes are the smallest units of speech that serve to distinguish words from each other. Each phoneme has a unique articulatory and distinguishable acoustic characteristic. In combination with other phonemes, they can form larger units such as syllables and words. For example, the words ”pin” and ”bin” differs with the phoneme /p/ and /b/, giving the words completely different meanings. Also, the words ”bin” and ”bean” sound similar but have different meanings since the sound between the letter b and n are different by the phonemes /i/ and /ea/.

The acoustic characteristics of a given phoneme change based on its immediate phonetic environment. The phonetic environment refers to having various anatomical structures (lips, tongue, and vocal cords) and the degree of effort put into making the sound. For example, the phoneme /t/ has different acoustic characteristics in different words, e.g., in ”tea”, ”tree”, ”city”, ”beaten” and ”steep” [7].

2.1.1 Context-independent phonemes

One can model the acoustic realization of words with phoneme models. Context-independent (CI) phonemes are modeled to be independent of their neighboring phonetic context. Monophones are such CI phonemes and represent the acoustic parameters of a single phoneme. For a language with only N phonemes, only N unit instances are necessary. A disadvantage of using CI phones is that they do not model co-articulation, providing a lower accuracies in speech recognition.

2.1.2 Context-dependent phonemes

The acoustic realization of words can also be modeled with context-dependent (CD) phoneme models. CD phonemes can improve speech recognition accuracy significantly, given that there are enough training data to estimate the CD parameters. Here, the context is limited to its left and right neighboring phonemes, only the immediate left and right phonetic context matters. Triphones are such CD phonemes that depend on their neighboring phones. At most, there are N^3 units instances; however, the number of unit instances is usually much lower.

A disadvantage of using context-dependent phones is that they provide an excessive amount of model parameters in speech recognition. Thus, training thousands of triphone units can be a complicated and time-consuming procedure [7].

2.2 Automatic speech recognition

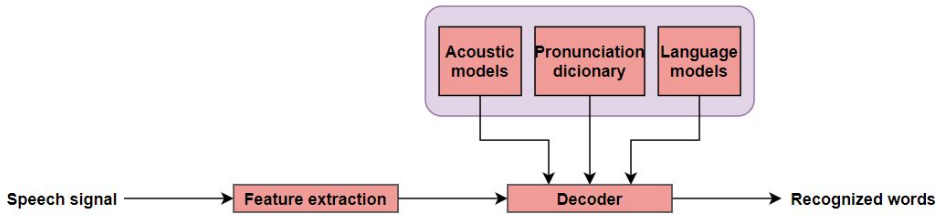


Figure 2.1: Architecture for automatic speech recognition.

Automatic speech recognition (ASR) is the process of automatically converting acoustic signals of a speech utterance into text transcription. The overall structure for ASR is illustrated in Figure 2.1. The most likely spoken words are determined based on the given speech signal. They are achievable by comparing a set of parameters describing the speech signal with trained acoustic model parameters. A trained acoustic model predicts either words or phones.

Speech recognition is stated as follows. Given a sequence of acoustic feature vectors (observations) $\mathbf{O} = o_1, o_2, ..., o_n$ and a word sequence $\mathbf{W} = w_1, w_2, ..., w_m$, the most likely word sequence $\mathbf{W*}$ is given by

$$\mathbf{W*} = \arg\max_{\mathbf{W}} P(\mathbf{W}|\mathbf{O}) \tag{2.1}$$

Applying Bayes' Theorem to Eq. 2.1 gives

$$P(\mathbf{W}|\mathbf{O}) = \frac{P(\mathbf{O}|\mathbf{W})P(\mathbf{W})}{P(\mathbf{O})}, \tag{2.2}$$

where $P(\mathbf{W})$ is the probability of an uttered word W and the conditional probability P($\mathbf{O}|\mathbf{W}$) computes the likelihood of observation **O** given word sequence **W**. $P(\mathbf{O})$ is the probability that the observation O will occur. Since variable **O** is already fixed, Equation (2.1) is reduced to

$$\mathbf{W*} = \arg\max_{\mathbf{W}} P(\mathbf{O}|\mathbf{W})P(\mathbf{W}) \tag{2.3}$$

$P(\mathbf{O}|\mathbf{W})$ also represents an acoustic model (AM) and $P(\mathbf{W})$ a language model (LM). The language model provides the a priori probability that a sequence of words **W** is uttered. This thesis focuses mainly on the acoustic model.

2.3 Feature extraction

Feature extraction is performed to provide a compact representation of the speech waveform. The process converts speech signals into sequences of acoustic vectors $\mathbf{O} = \{o_1, o_2, ..., o_N\}$. Afterward, the acoustic vectors are used as input to an acoustic model.

There are several feature extraction techniques. Two feature extraction techniques discussed in this chapter include Mel-frequency cepstral coefficients (MFCCs) and Feature-space maximum likelihood linear regression (fMLLR).

2.3.1 Mel-frequency cepstral coefficients

Mel-frequency cepstral coefficients (MFCCs) are typical feature vectors in speech recognition.

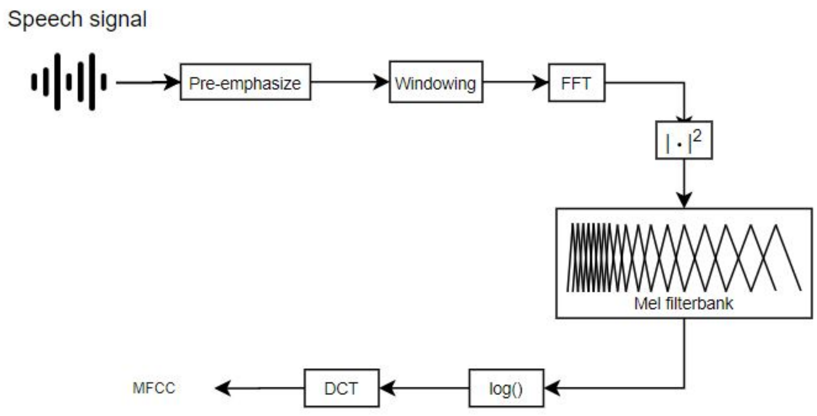


Figure 2.2: Process of generating MFCCs.

MFCCs are generated in several steps, as illustrated in Figure 2.2. Firstly, the audio signal is pre-emphasized and parted into frames. The next step calculates each frame's spectral coefficients using a fast Fourier transform (FFT) and is further sent to a Mel filterbank to filter the spectrum of speech signals using triangular bandpass filters. The Mel filterbank indicates how much energy occurs at various frequency regions. Lastly, discrete cosine transformation (DCT) is applied to the logarithm of the filterbank energies. Hence, the cepstral features for each frame are obtained. Typically the first 13 cepstral coefficients are used in speech recognition as input features to an acoustic model.

The time derivatives (deltas) and accelerations (delta deltas) improves the robustness of the recognition task. These features are concatenated to the original cepstral features, thus providing a 39-dimensional MFCC feature vector for each frame [13].

Feature normalization

A popular data preprocessing technique is to take the per-sample feature normalization of the extracted features. For example, the cepstral mean normalization (CMN) technique subtracts the per-utterance mean μ_i of MFCC features in order to reduce acoustic channel distortion. The CMN is computed by first estimating the per-utterance mean

$$\bar{\mu}_i = \frac{1}{T}\sum_{t=1}^{T} o_i^t, \tag{2.4}$$

for dimension i and T total number of frames in the utterance, and then the mean is subtracted from all frames in the utterance as

$$\bar{o}_i^t = o_i^t - \bar{\mu}_i. \tag{2.5}$$

2.3.2 Feature-space maximum likelihood linear regression

Feature-space maximum likelihood linear regression (fMLLR) is a feature adaption technique that deals with speaker variability. The speaker variability is reduced through the estimation of a feature transformation matrix. The goal of fMLLR is to normalize features to fit the speaker better.

In fMLLR, feature-space transformations are performed, where the features $\mathbf{o}(t)$ are transformed directly according to

$$\hat{\mathbf{o}}^{(s)}(\tau) = \mathbf{A}^{(s)}\mathbf{o}(\tau) + \mathbf{b}^{(s)} = \mathbf{W}^{(s)}\xi(\tau) \tag{2.6}$$

where

$$\mathbf{W}^{(s)} = \left[\mathbf{A}^{(s)}, \mathbf{b}^{(s)}\right], \tag{2.7}$$

$\mathbf{W}^{(\mathbf{s})}$ represents the transformation matrix and ξ(t)=$[\mathbf{o}_t^T, 1]^T$ is the extended feature vector. Matrices $\mathbf{A}^{(s)}$ and $\mathbf{B}^{(s)}$ are estimated iteratively and have to be initialized, e.g. as a diagonal matrix with ones on the diagonal and a zero vector, respectively [4].

A linear transform is applied to the feature vectors for every frame, where the transform parameters are estimated by optimizing the following auxiliary function

$$\begin{aligned}\mathbf{Q}(\mathbf{M},\mathbf{M}) = K - \frac{1}{2}\sum_{s=1}^{S}\sum_{m=1}^{M}\sum_{\tau=1}^{T^{(s)}}\gamma_m(\tau)\Big[K^{(m)} + \log(|\mathbf{\Sigma}^{(m)}|) - 2\log(|\mathbf{A}^{(s)}|) \\ +(\mathbf{A}^{(s)}\mathbf{o}(\tau)+\mathbf{b}^{(s)}-\mu^{(m)})^T\mathbf{\Sigma}^{(m)-1}(\mathbf{A}^{(s)}\mathbf{o}(\tau)+\mathbf{b}^{(s)}-\mu^{(m)})\Big],\end{aligned} \tag{2.8}$$

for M total number of components associated with the particular transform and normalization constant K^m associated with Gaussian component m. The transformed mean and variance for component m are given by

$$\hat{\mu}^{(m)} = \frac{\sum_{s=1}^{S}\sum_{\tau=1}^{T^{(s)}}\gamma_m(\tau)(\hat{\mathbf{o}}^{(s)}(\tau)-\hat{\mu}^{(m)})(\hat{\mathbf{o}}^{(s)}(\tau)-\hat{\mu}^{(m)})^T}{\sum_{s=1}^{S}\sum_{\tau=1}^{T^{(s)}}\gamma_m(\tau)} \tag{2.9}$$

and

$$\hat{\mathbf{\Sigma}}^{(m)} = \frac{\sum_{s=1}^{S}\sum_{\tau=1}^{T^{(s)}}\gamma_m(\tau)(\hat{\mathbf{o}}^{(s)}(\tau)-\hat{\mu}^{(m)})(\hat{\mathbf{o}}^{(s)}(\tau)-\hat{\mu}^{(m)})^T}{\sum_{s=1}^{S}\sum_{\tau=1}^{T^{(s)}}\gamma_m(\tau)} \tag{2.10}$$

at time τ, respectively.

For deep neural networks (DNNs), fMLLRs are optimized to maximize the cross-entropy using backpropagation. This discriminative criterion is referred to as feature-space discriminative linear regression (fDLR). The transformation is applied to either each input vector in the DNN or to individual frames before concatenation [19].

2.4 Acoustic models

This section deals with the quantity $P(\mathbf{O}|\mathbf{W})$. The acoustic model (AM) in ASR is about modeling a sequence of feature vectors (observations) given a sequence of words.

2.4.1 Hidden Markov models

A hidden Markov model (HMM) is a commonly used acoustic model in ASR. Each word is represented as a sequence of phonetic units, and each unit is represented by an HMM containing a predefined number of states.

An HMM augments the Markov chain. A Markov chain computes the probability of a sequence of observable events. A first-order Markov chain assumes that when predicting the current state j at a given time, it is only dependent on the previous state i. A second-order Markov chain assumes that the two previous states and the current state matters when predicting the next state. HMMs differ in that they observe states indirectly, also referred to as hidden states.

The following parameters characterize HMMs:

Q = $q_1, q_2, ..., q_N$ - State sequence containing N states.

A = $a_{11}, ..., a_{ij}, ..., a_{NN}$ - a transition probability A where each a_{ij} represents the probability of moving from state i to state j.

O = $o_1, o_2, ..., o_T$ - observation sequence with T observations.

B = $b_i(o_t)$ - emission probability, a sequence of observation likelihoods. Each probability expresses the probability of an observation o_t being generated from a state i.

$\pi = \pi_1, \pi_2, ..., \pi_N$ - initial probability distribution over states where π_i is the probability that the Markov chain will start in state i.

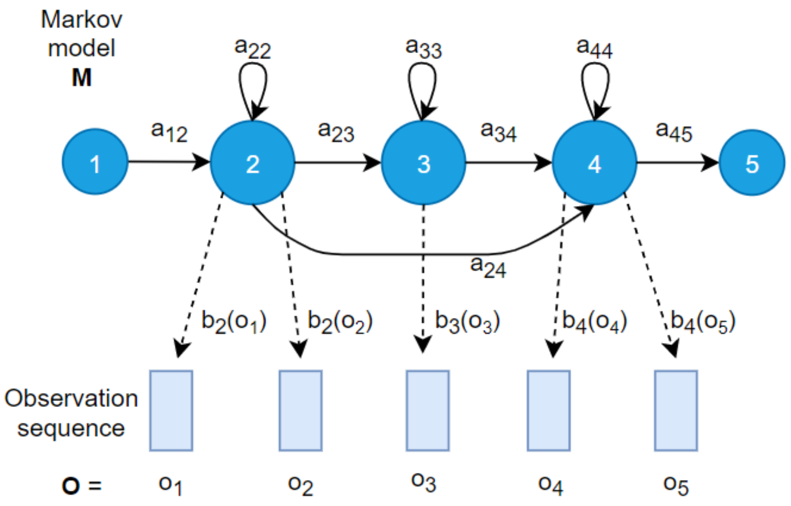


Figure 2.3: Example of a HMM with 3 states.

Figure 2.3 illustrates a HMM with 3 states. The figure provides a visualization of the mentioned characterized parameters for the HMM.

Like a first-order Markov chain, the probability of a state depends only on the previous state in a first-order HMM. The formal definition of dependency follows.

$$P(q_i|q_1...q_{i-1}) = P(q_i|q_{i-1}) \tag{2.11}$$

Additionally, in terms of computing the probability of an output observation o_i of a HMM, only the state that produced the observation matters and not any other states or observations. More formally,

$$P(o_i|q_1...q_i, ..., q_T, o_1, ..., o_i, ..., o_T) = P(o_i|q_i) \tag{2.12}$$

Overall an HMM deals with the three following fundamental problems [8].

Problem 1 (Likelihood)

Given an HMM λ = (A,B) and an observation sequence **O**, determine the likelihood $P(\mathbf{O}|\lambda)$.

An HMM with N hidden states and an observation sequence of T observations have a complexity of N^T. The complexity can be a large number. Therefore, it is more feasible to compute the total observation likelihood through a forward algorithm. The forward algorithms sums the probability over all possible hidden state paths that could generate the observation sequence. The algorithm efficiently calculates

$$\alpha_t(j) = P(o_1, o_2 ... o_t, q_t = j|\lambda) \tag{2.13}$$

recursively at time t given current state j.

For a given state q_j at time t, the value $\alpha_t(j)$ is computed as

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t) \tag{2.14}$$

In Equation 2.14, $\alpha_{t-1}(i)$ is the previous forward path probability, a_{ij} is the transition probability from previous state q_i to current state q_j, and $b_j(o_t)$ the state observation likelihood of the observation symbol o_t given current state j. This gives,

$$P(\mathbf{O}|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{2.15}$$

Problem 2 (Decoding)

Decoding deals with finding the most probable sequences of states Q = $q_1, q_2, q_3, ..., q_T$, given an HMM λ = (A,B) and a sequence of observations $O = o_1, o_2, ..., o_T$.

The most common decoding algorithm for HMMs is the Viterbi algorithm. The Viterbi algorithm takes the most probable path over the previous path probabilities.

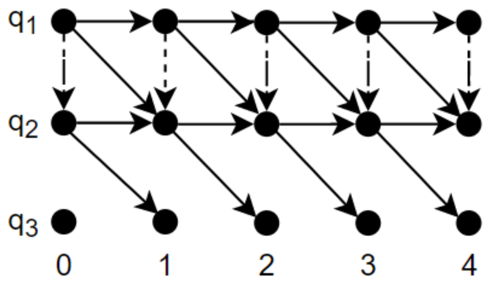


Figure 2.4: HMM trellis.

Figure 2.4 illustrates an HMM trellis, where the Viterbi trellis processes the observation sequence left to right, filling out the trellis. Each cell of the trellis, $\nu_t(j)$, represents the probability of being in state j after seeing the first observations and passing through the most probable state sequences $q1, ..., q_{t-1}$, given λ.

$$\nu_t(j) = \max_{q_1,...q_{t-1}} P(q_1...q_{t-1}, o_1, o_2...o_t, q_t = j|\lambda) \tag{2.16}$$

The value of each cell $\nu_t(j)$ is computed by recursively taking the most probable path that could lead to this cell.

$$v_t(j) = \max_{i=1}^{N} \nu_{t-1}(i) a_{ij} b_j(o_t) \tag{2.17}$$

In Equation 2.17, $\nu_{t-1}(i)$ is the previous Viterbi path probability from the previous time step and $b_j(o_t)$ the state observation likelihood of the observation symbol o_t given the current state j.

The Viterbi algorithm also has back pointers. By keeping track of the path of hidden states that led to each state, and then returning the best path to the beginning, one can obtain the best state sequence.

Problem 3 (Learning)

The goal of training an HMM is to learn the HMM parameters A and B, given an observation sequence **O** and the set of states in the HMM. One can solve the problem by using the forward-backward algorithm.

The forward-backward algorithm, or Baum-Welch algorithm, trains on both the transition probabilities A and the emission probabilities B of the HMM. It is an iterative algorithm that computes an initial estimate for the probabilities, then uses those estimates to compute even better estimates. This way, the probabilities are improving iteratively as the model learns.

Given state i at time t, the backward probability β is the probability of seeing the observations from time t+1 to the end.

$$\beta_t(i) = P(o_{t+1}, o_{t+2}...o_T|q_t = i, \lambda) \tag{2.18}$$

The backward probability processes are

1. Initialization:

$$\beta_T(i) = 1, \quad 1 \leq i \leq N \tag{2.19}$$

2. Recursion:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j), \quad 1 \leq i \leq N,\ 1 \leq t < T \tag{2.20}$$

3. Termination:

$$P(O|\lambda) = \sum_{j=1}^{N} \pi_j b_j(o_1)\beta_1(j) \tag{2.21}$$

The forward and backward probabilities are used to compute the transition probability a_{ij} and the observation probability $b_i(o_t)$ from an observation sequence.

To learn the HMM model, the probability $\gamma_t(j)$ of being in state j at time t, and the probability ξ_t of being in state i at time t and state j at time t+1 given the observation sequence, is used to estimate A and B, where

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} \tag{2.22}$$

and

$$\xi_t i, j = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} \tag{2.23}$$

The total expected number of transitions from state i is found by summing over all transitions out of state i

$$\hat{a_{ij}} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1}\sum_{k=1}^{N} \xi_t(i,k)} \tag{2.24}$$

and the percentage of times that one were in state j and saw symbol v_k is defined by

$$\hat{b}_j(v_k) = \frac{\sum_{t=1}^{T} 1_{o_t=v_k}\gamma_i(t)}{\sum_{t=1}^{T} \gamma_i(t)} \tag{2.25}$$

Consequently, assuming that the previous estimate of A and B is found, the transition A and observation B from an observation sequence O can be re-estimated using Equation 2.24 and 2.25.

To sum up, the forward-backward algorithm starts with some initial estimate of the HMM parameters λ = (A, B). Then two steps are iteratively run. Because the forward-backward algorithm is a special case of the Expectation-Maximization (EM) algorithm, the expectation step computed in our case is the expected state occupancy count γ and the expected state transition count ξ from preceding A and B probabilities. The M-step used γ and ξ to recompute the new A and B probabilities [8].

2.4.2 Hidden Markov models use for speech recognition

Left-to-right HMM

A common HMM-topology is a left-to-right HMM.

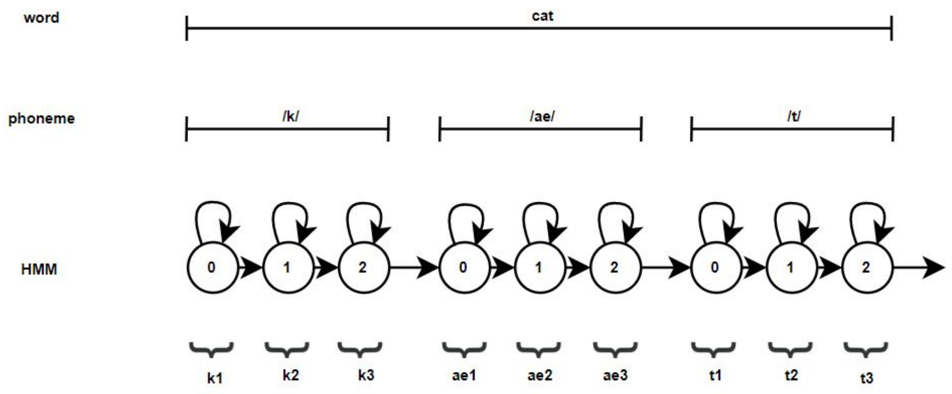


Figure 2.5: Left-to-right HMM wit three states for phonemes.

Figure 2.5 illustrates the left-to-right HMM with three states, where each state represents a sub phone. There are three states for each phoneme because a phoneme is not stationary. The first and the last part of a phoneme are typically different from the middle part. The states 0, 1, and 2 correspond to the beginning, middle, and end of a phoneme, respectively. Moreover, word and sentence HMMs are constructed by concatenating these phoneme-level HMMs.

Decision tree clustering

The CD phonemes could hold a large number of units. In such cases, the CD units can be clustered into a smaller set whose distribution is robustly estimated using decision trees. Additionally, contexts with little data are combined until sufficient data are available. Clustering can occur at a phoneme level or a state level.

A decision tree is a binary tree, where yes/no phonetic questions are attached to each node. Initially, all states are placed at the root of the tree. Based on the answer to the phonetic question, the states are split and continue to split until the states have reached the leaf-nodes.

The decision trees are built on a top-down sequential optimization process. The phonetic questions are chosen based on which split of the root node gives the best split. This process will be repeated until the increase in the log-likelihood falls below a specified threshold. The decrease in the log-likelihood is calculated for the merging terminal nodes with different parents. If the decrease in log-likelihood is smaller than the threshold, the splitting process is stopped and the leaf nodes will then be merged.

2.4.3 Gaussian mixture model

The output probability density function for each state of an HMM, or the emission probability $b_i(o_t)$, can be modeled by a Gaussian mixture model (GMM). A GMM is an acoustic model, and is as a weighted sum of single Gaussian models of different means and covariances.

A continuous random observation o has a Gaussian mixture distribution if its probability density function (PDF) is defined by

$$
\begin{aligned}
p(o) &= \sum_{m=1}^{M} \frac{c_m}{\sqrt{2\pi}\sigma_m} \exp\left[-\frac{1}{2}\left(\frac{o-\mu_m}{\sigma_m}\right)^2\right] \\
&= \sum_{m=1}^{M} c_m \mathcal{N}(o;\mu_m,\sigma_m^2), \\
&(-\infty < o < \infty; \sigma_m > 0; c_m > 0)
\end{aligned}
\tag{2.26}
$$

The PDF of a multivariate D-dimensional Gaussian mixture distribution is defined by

$$
\begin{aligned}
p(\mathbf{o}) &= \sum_{m=1}^{M} \frac{c_m}{(2\pi)^{\frac{D}{2}} |\mathbf{\Sigma_m}|^{\frac{1}{2}}} \exp\left[-\frac{1}{2}(\mathbf{o}-\mu_m)^T \Sigma_m^{-1} (\mathbf{o}-\mu_m)\right] \\
&= \sum_{m=1}^{M} c_m \mathcal{N}(\mathbf{o};\mu_m,\mathbf{\Sigma}_m), \qquad (c_m > 0)
\end{aligned}
\tag{2.27}
$$

of a D-dimensional observation vector $\mathbf{o}$, with M being the total number of mixture components, and c_m, μ_m and $\mathbf{\Sigma}_m$ are the weighting factor, mean vector and covariance matrix of the m normal component of a state respectively. Additionally, the positive mixture weights satisfy $\sum_{m=1}^{M} c_m = 1$ [19].

Forced alignment

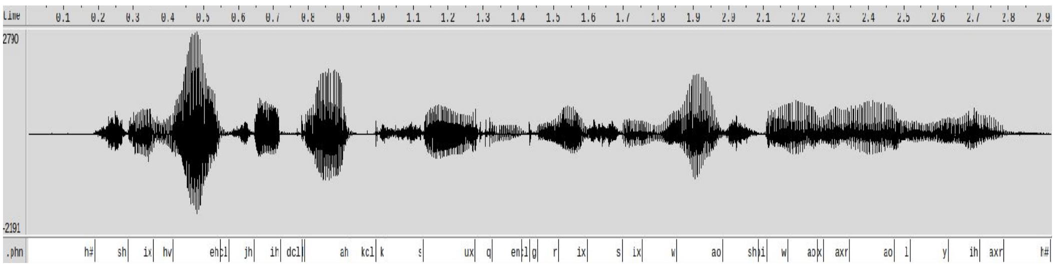


Figure 2.6: Waveform of spoken sentence ”She had your dark suit in greasy wash water all year”, from the TIMIT Corpus dataset.

Forced alignment is the process of aligning a known sequence of phonemes from a transcription to the corresponding audio recording. Fig. 2.6 displays an audio waveform and its corresponding transcription data ”She had your dark suit in greasy wash water all year” from the TIMIT Corpus dataset [3]. One can think of the alignments as audio with time-stamps, in which the time-stamps correspond to spoken phonemes in the audio.

In ASR, through forced alignment, a GMM-HMM assigns an HMM to state to each corresponding frame. The Viterbi algorithm in the HMM outputs the most probable observed sequence given the input speech signals. The alignments are further used as labels for training an acoustic model.

2.4.4 Deep neural networks

Deep neural networks (DNNs) have been shown to outperform GMMs as acoustic models in speech recognition over the last few years [6].

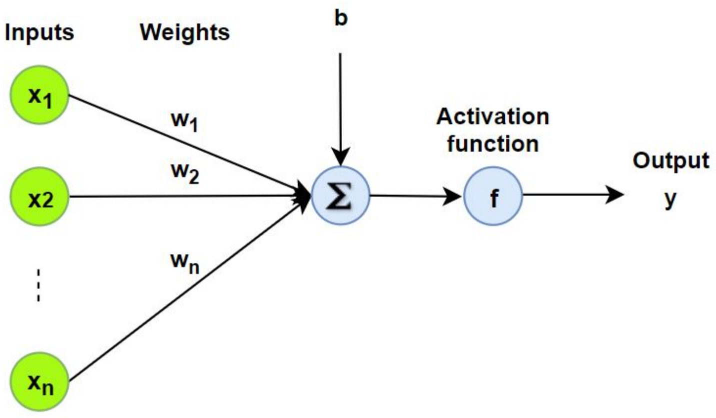


Figure 2.7: Perceptron architecture.

Figure 2.7 illustrates a perceptron, a building block for the DNN. The figure shows inputs $\{x_1, x_2, ..., x_n\}$ from the input layer generating an output y at the output node. The perceptron sums all the weights $\{w_1, w_2, ..., w_n\}$ from the previous layer (l-1), adds a bias b and applies an activation function f to the weighted sum. The mapping of the inputs x_i from the previous layer to the output y is defined by

$$y = f(b + \sum_i w_i x_i) = f(z), \tag{2.28}$$

where f is the predefined activation function. An activation function is a non-linear function, typically tanh function $f(z) = (exp(2z) - 1)/(exp(2z + 1)$, the sigmoid function $f(z) = 1/(1 + exp(-z))$ or a rectified linear unit (ReLU) $f(z) = max(0, z)$ [10].

In multi-class classification tasks where there are k distinctive classes, an output unit j converts the total inputs, x_j, into a class probability, p_j through a non-linear softmax function defined by

$$p_j = \frac{exp(x_j)}{\sum_k exp(x_k)} \tag{2.29}$$

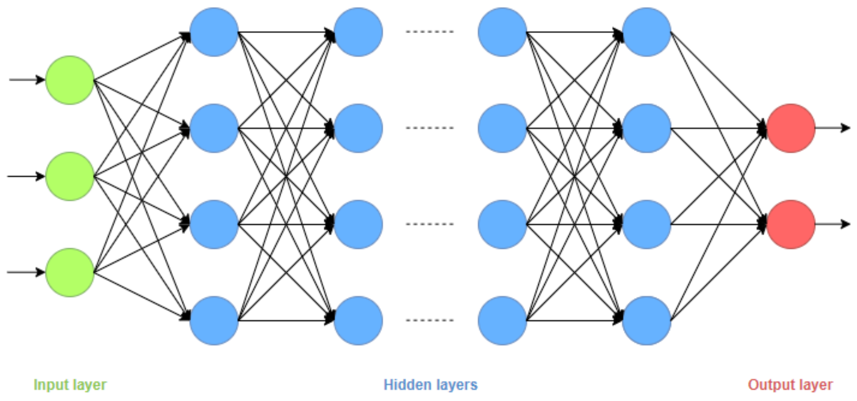


Figure 2.8: A deep neural network model.

A DNN is an artificial neural network with more than one layer of hidden units between its inputs and its outputs, as shown in Figure 2.8. A Multilayer Perceptron (MLP) is often used as a baseline DNN. MLP consists of two or more hidden layers and is a feed-forward neural network where all the neurons in one layer are fully connected to the neurons in the adjacent layer.

DNNs can be discriminatively trained by backpropagation. After each forward pass through the network, backpropagation performs a backward pass while adjusting the weights and the biases. The backpropagation aims to minimize the cost function that measures the difference between the target outputs, and the actual predicted output is performed. The cost function C is a cross-entropy between the target probabilities, d, and the outputs of the softmax, p,

$$C = -\sum_j d_j log p_j \tag{2.30}$$

where the target probabilities are the class labels provided to train the DNN classifier [6].

Overfitting

DNNs as acoustic models are prone to overfitting. Overfitting is when the DNN does not generalize well on new data and can be prevented by using regularization techniques. Conventional regularization techniques are dropout, early stopping, and L1 and L2 regularization. Dropout works by temporarily ignoring a given number of units in a layer of the neural network given a dropout rate p. For early stopping, the training of a DNN is stopped when the model does not improve on the validation set for a specific number of epochs. Lastly, L1 and L2 regularization adds a regularization term to prevent the coefficients from fitting so perfectly [14].

2.4.5 Performance evaluation

The performance of an ASR system can be evaluated on frame-by-frame recognition or phoneme recognition. A frame-by-frame phoneme classifier measures either the state level accuracy of a phoneme or a phoneme level accuracy. The classifier returns a score indicating the confidence that the predicted frame is correctly matched against the reference frame, a target. The best candidates have the highest scores. In contrast, phoneme recognition identifies individual phonemes in a sentence. The phonemes, in such cases, spans over several frames. In this work, the focus is mainly on evaluation of frame-by-frame recognition.

Concerning the frame-by-frame recognition, there are several methods for calculating the prediction accuracy.

Phone error rate

The phoneme error rate (PER), or phone accuracy rate, is the most common evaluation metric in ASR. The formula of PER is based on the following expressions

· Substitution (S) = number of substitutions
· Insertion (I) = number of insertions
· Deletion (D) = number of deletions

$$PER = \frac{S + D + I}{N} \times 100, \tag{2.31}$$

where N is the total number of labels in the reference utterance. The phone accuracy rate is calculated by

$$\text{Accuracy} = 100 - \text{PER} \tag{2.32}$$

Classification accuracy

Another metric is the classification accuracy. It is the ratio of the number of correct predictions to the total number of input samples.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

F1 score

Additionally, F1 score is a weighted average of the precision and recall, where the F1 score reaches its best score at 1 and the worst score at 0. The formula for the F1 score is defined by

$$\text{F1} = 2 \times \frac{(\text{precision} \times \text{recall})}{(\text{precision} + \text{recall})} \tag{2.33}$$

Confusion matrix

Another conventional method is using a confusion matrix (CM). The CM is a table that depicts the performance of a classifier on correctly classified and misclassified labels. It is the counts of the classified and misclassified labels that fill the table. Each row of a CM represents the predicted classes, and each column represents the actual classes (the opposite is also correct).

2.5 Semi-supervised learning

Semi-supervised learning (SSL) combines supervised and unsupervised learning techniques. Supervised learning trains an artificial neural network on the labeled data, while unsupervised learning trains the ANN on the unlabeled data. Hence, SSL methods use both labeled and unlabeled data to improve learning performance. SSL is mostly used when the labeled data are scarce.

2.5.1 Teacher-student training

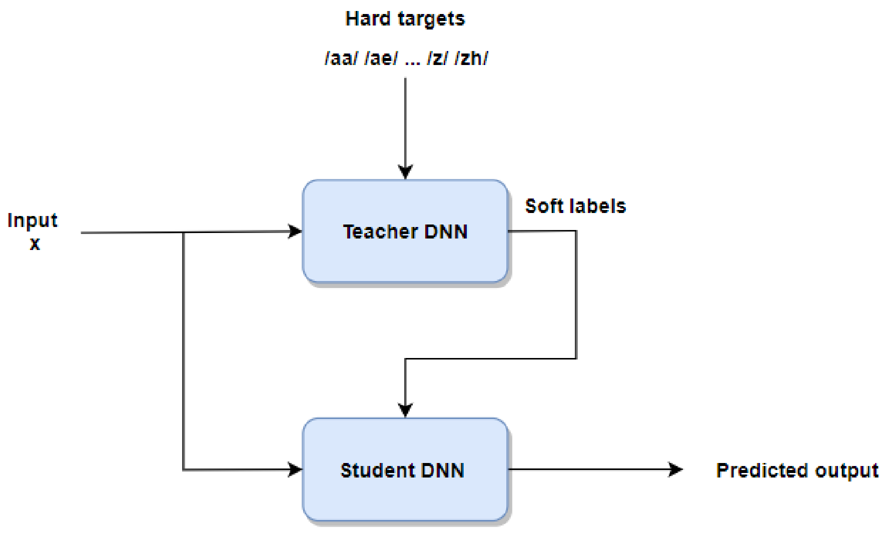


Figure 2.9: Teacher-student network.

A possible method of implementing SSL is by using teacher-student (T/S) network, as illustrated in Fig. 2.9.

The teacher network is an artificial neural network (ANN) that has already been trained on labeled data achieved from forced alignment, also called hard targets [16] and is later used to produce labels on the unlabeled data. The network outputs class probabilities, also referred to as soft labels. The student network tries to mimic the teacher network's behavior by trying to replicate its outputs at every time step. The student network uses the teacher model's soft labels as its targets.

T/S learning aims to minimise the Kullback-Leibler (KL)-divergence between the output distribution of the teacher model $P_T(q|x)$ and the student model $P_S(q|x)$

$$KL(P_T||P_S) = \sum_t \sum_{i=1}^{N} P_T(q_i|x_t) log\left(\frac{P_T(q_i|x_t)}{P_S(q_i|x_t)}\right), \tag{2.34}$$

for i frame index, N total number of HMM states, the i-th shared state q and x_t input vector at time t [11]. Minimizing the KL-divergence is equivalent to minimizing the loss function between P_T and P_S

$$L = -\sum_t \sum_{i=1}^{N} P_T(q_i|x_t) log P_S(q_i|x_t). \tag{2.35}$$

Chapter 3

Related literature

This chapter provides a brief literature review on state-of-the-art methods performed on ASR systems. Section 3.1 investigates research made on T/S networks, Section 3.2 introduces two state-of-the-art acoustic models, and lastly, Section 3.3 introduces a research paper that experimented with a toolkit to reduce PER.

3.1 Teacher/student learning

Cambridge University and Li et al. (2017) released a paper regarding experimental studies on teacher/student (T/S) training of DNN acoustic models [11]. The experiment was performed on the TIMIT speech corpus. In the experiment, the student models were restricted by both model complexity and teacher performance. The student model was trained from a fully-connected 7-layer teacher model. A PER of 25.76% was achieved from the student baseline. The paper also examines training on recurrent neural networks (RNN) and ensemble learning, and achieved lowest PER of 23.73% using ensemble learning.

Kim et al. (2017) proposed using T/S learning to transfer knowledge from a large speech recognition model to an end-to-end online model. An offline end-to-end model as a teacher model was made by a deep bidirectional RNN with LSTM units (BLSTM) to predict the correct label sequence given the entire utterance. Once the model was trained, the knowledge was transferred to the student model, an LSTM-KL model that could operate online without access to the future input frames. The proposed model was shown to outperform models that are trained from random initialization [9].

In 2019, another research paper experimented with teacher models trained on large audio databases, and student models were trained on a small-sized audio database. Li et al. (2019) proposed using a cross-modal T/S training framework, where the teacher and student audio models consisted of DNNs. The DNN proposed in this paper was composed of six hidden layers and 1024 sigmoid units in each layer, trained on fMLLR features. The proposed solutions reduced the PER from

26.7% to 21.3%. The improvement was partly from using a more extensive training set, covering a broader acoustic space [12].

3.2 State-of-the-art acoustic models

3.2.1 Reccurent neural networks

Recurrent neural networks (RNNs) can be used as acoustic models in ASR. The advantage of using RNNs is that they can capture temporal information and learn short and long-term speech dependencies. A popular type of RNN, are long-short memory networks (LSTMs), which rely on memory cells that are controlled by forget, input, and output gates. The research paper conducted by Ravanelli et al. (2018) proposed a simplified architecture of Gated Recurrent Units (GRUs), called Light GRU (Li-GRU). GRUs simplified the complex LSTM cell design. The gating mechanism controlled the flow of information through various time-steps better. In Li-GRUs, the reset gate is removed, and ReLU activation functions implemented. Batch normalization is also used. Li-GRU reduced the per-epoch training time by more than 30% and improved recognition accuracies across different ASR paradigms.

3.2.2 Convolutional neural networks

Utilizing deep convolutional neural networks (CNN) over the commonly used DNNs on speech tasks was investigated by Tóth (2015). The CNNs are ANNs which detect features that are local in frequency and also tolerates small shifts in their positions. In this paper, the CNN was turned into a hieratical model which extended the locality to the time axis. The hieratical model trains another network on some posterior estimates. The paper also experimented with using maxout activation functions in the CNNs, which turned out to outperformed the commonly used ReLU and sigmoid functions. The proposed CNN provided a phoneme error rate reduction of 4.3% over ReLU CNNs. Additionally, it was also showed that adding dropout to the CNN also contributed to a lower phoneme error rate. [18].

3.3 Toolkits

There are various software available for performing speech recognition tasks. Open-source softwares such as the HTK, CMU Sphinx, and Kaldi toolkit are popular choices. In 2019, Ravanelli et al. (2019) experimented with the Kaldi toolkit and the Pytorch framework in the Python language that builds neural networks, and the Pytorch-Kaldi project was created. Results confirmed that the Pytorch-Kaldi toolkit could be effectively used to develop modern state-of-the-art speech recognizers. The toolkit gave a phone error rate (PER) of 13.8% on the TIMIT speech corpus and is amongst the lowest error rates dated so far on the TIMIT corpus [17].

Chapter 4

Method

In this chapter, the implemented and tested methods completed in our work are presented. Section 4.1 describes the overall ASR system implemented, and Section 4.2 provides further details on the acoustic models used in this work.

4.1 ASR system

The semi-supervised learning model, a teacher-student (T/S) network, was implemented. In this work, the ASR system was parted into two parts: a teacher model trained on labeled data and a student model trained on unlabeled data. First off, the teacher model was built on a DNN. After, the student model was built on a separate yet same DNN architecture as the teacher model.

When training the DNNs, the teacher model was trained on hard targets which were generated by GMM-HMM through forced alignment, and the student model was trained on soft targets obtained from the output of the teacher model. The hard targets were state posteriors computed on the speech data in the teacher model. This way, the student model will not be trained on labeled data and, at the same time, receives gains from the trained teacher model.

The teacher and the student models were trained on the full training set with MFCC features and fMLLR features.

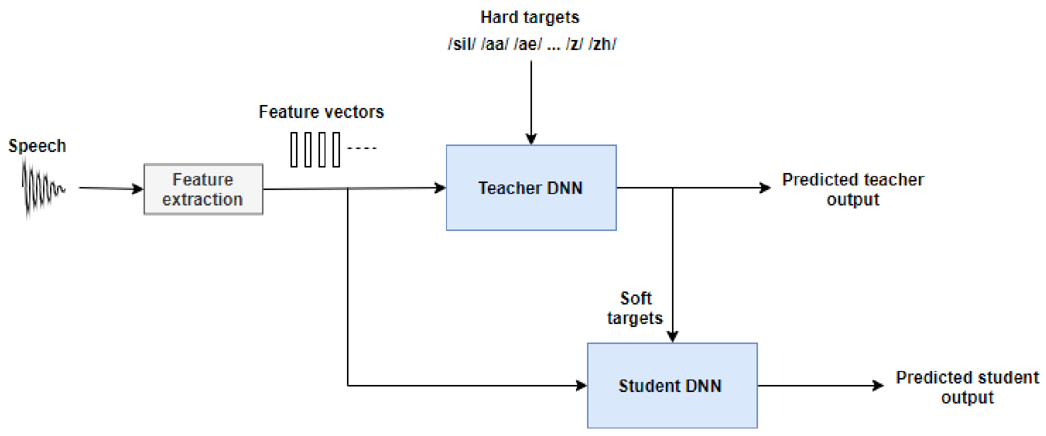


Figure 4.1: Teacher-student network.

The structure of the teacher-student network is illustrated in Figure 4.1.

4.2 Classifier

The purpose of the DNN is to act as a classifier where it classifies states frame-by-frame. From there, the aim is to perform phoneme recognition on given utterances. Evaluation was performed on two DNN models, a monophone model and a triphone model. Therefore, two teacher-student networks were implemented, one network built on the monophone models and the other on the triphone models.

4.2.1 Input features

In this experiment, 13 MFCC coefficients and 40-dimensional fMLLR feature vectors were extracted for each frame from the speech waveforms. Before feeding the features to the DNNs, preprocessing using cepstral mean and variance normalization was performed on the features.

4.2.2 Targets

For frame-by-frame phoneme classification, the predictions made by the DNN were compared to targets for every frame. The targets were a list of states, one state for each frame in the utterance. The frames predicted by the DNN were compared to the target for the same frame. The targets are also known as the true labels of the network.

4.2.3 Evaluation

The teacher-student network was evaluated in three different forms. The first form evaluated the training, test, and validation set with frame-by-frame classification accuracies, checking which frames were classified correctly on the data set. Subsequently, when the predicted states made by the classifier were obtained, they were further mapped into phonemes. The final evaluation form used a CM to visualize which phonemes were classified the most and to see what kind of errors the models make.

Chapter 5

The TIMIT Corpus

5.1 Database

The experiments conducted in this paper were evaluated on the TIMIT Corpus data set. The TIMIT speech corpus is an acoustic-phonetic speech corpus developed by Texas Instruments (TI), SRI International (SRI), and Massachusetts Institute Of Technology (MIT) to provide speech data for development and evaluation of ASR systems. TIMIT contains recordings of 630 speakers of eight major dialects of American English, each reading ten phonetically rich sentences. The corpus includes time-aligned phonetic and word transcriptions and a 16-bit, 16kHz speech waveform file for each utterance [5].

Moreover, there are 44 phones in the English language. However, the TIMIT corpus has a total of 48 phonemes, including the speech units silence and closure as phonemes.

Table 5.1: TIMIT corpus 48-phoneme set.

Speech unit	
Phoneme	/aa/, /ae/, /ah/, /ao/, /aw/, /ax/, /ay/, /b/, /ch/, /d/, /dh/, /dx/, /eh/, /el/, /en/, /er/, /ey/, /f/, /g/, /hh/, /ih/, /ix/, /iy/, /jh/, /k/, /l/, /m/, /n/, /ng/, /ow/, /oy/, /p/, /r/, /s/, /sh/, /t/, /th/, /uh/, /uw/, /v/, /w/, /y/, /z/, /zh/
Silence	/sil/, /epi/
Closure	/cl/, /vcl/

Table 5.1 shows the phonemes found in the TIMIT corpus.

Table 5.2: Number of utterances, speakers and frames in the training set, validation set and test set in the TIMIT Corpus.

Spoken / Data set	Utterances	# speakers
Training set	3696	462
Validation set	400	50
Test set	192	24
Total data set	6300	630

The TIMIT corpus has a training set, a validation set, and a test set. The training set has 3696 utterances, the validation has 400 utterances, and the test set has 192 utterances, as shown in Table 5.2.

For this experiment, the original phoneme set in TIMIT was used for evaluation and not the reduced set that is commonly used.

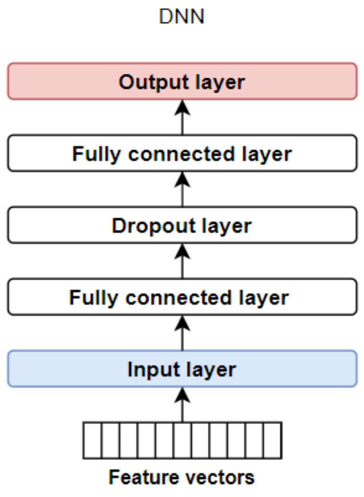
Chapter 6

Experiment

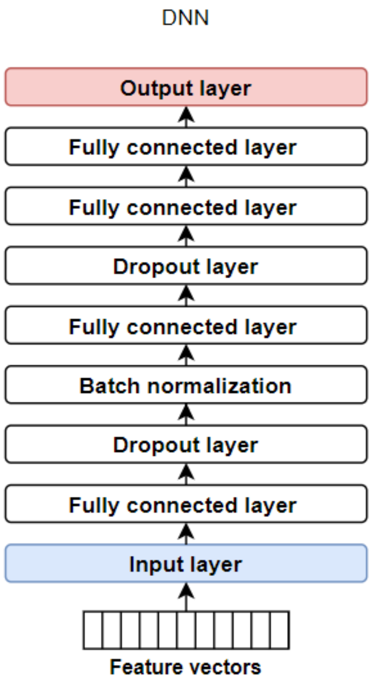
This chapter gives a general description of the experiments performed with the TIMIT corpus in this work. Section 6.1 gives details on the system architecture and the actual parameters utilized to build the networks in this experiment, and Section 6.2 presents the toolkits employed in the experiments and how they were used.

6.1 DNN

The teacher-student (T/S) networks were built on deep neural networks (DNNs). Two T/S networks were made, one on monophone models which were DNNs trained on monophone targets and the other on triphone models which were DNNs trained on triphone targets. The teacher model was trained on a completely labeled training set and the student model on completely unlabeled training set.



(a) Architecture of monophone model.



(b) Architecture of triphone model.

Figure 6.1: Architectures of DNNs.

Figure 6.1 shows the architecture of the two separately implemented monophone models and triphone models, respectively.

Table 6.1: Toplogy of monophone model.

Layer type	Nodes	Activation function
Input	256	ReLU
Fully connected	256	ReLU
Droput		
Fully connected	256	ReLU
Output	144	Softmax

Table 6.2: Toplogy of triphone model.

Layer type	Nodes	Activation function
Input	512	ReLU
Fully connected	512	ReLU
Droput		
Batch normalization		
Fully connected	256	ReLU
Droput		
Output	1896	Softmax

Table 6.1 and Table 6.2 illustrates the topology of the monophone models and triphone models. The DNNs in the teacher-student models were trained to classify the central frame of an 11-frame acoustic context window. The total number of frames in the training set was 1124823. The input layer had the same shape as the training data. For MFCCs, the shape was (11x13)=143, and for the fMLLRs, the input shape was (11x40)=440.

Moreover, the input layer of the monophone models was followed by a fully connected layer, a dropout layer, and another fully connected layer before the output layer. Each layer consisted of 256 nodes, and the dropout rate was set to 0.25. For the triphone models, the input layer was followed by four fully connected layers, with two dropout layers and a batch normalization between them. Each layer consisted of 512 neurons, and the dropout rate was also set to 0.25. The dropout and batch normalization was implemented to reduce overfitting that had occurred while training. Both models used ReLU activation functions in each layer, except for the output layer, where a softmax function was chosen as it is a common standard for classification tasks. All of these parameters were chosen based on trial and error, choosing the parameters that provided the highest frame accuracy. The trail and error experiments will not be reported in this paper.

The output layer of the monophone model consisted of 144 nodes and the triphone model of 1896 nodes. The numbers are chosen given the total amount of class labels in the targets. The monophones were obtained through a forced alignment using GMM-HMM with a left-to-right topology, giving a total of 144 states (3 states x 48 phonemes in TIMIT) as each phoneme had three corresponding states. Additionally, the triphones had in total 1896 states, where each phoneme had various amounts states.

The network was trained using the Adam optimizer and categorical cross-entropy as loss function, as these are commonly used parameters in classification tasks.

6.2 Implementation details

6.2.1 Kaldi

The experimental part of this work is based on the Kaldi toolkit [15]. Kaldi is an open-source toolkit used for speech recognition research. The toolkit's purpose is to be used by speech recognition researchers. The aim of the toolkit is to have a modern and flexible code that is easily understood, can be modified and extended. Kaldi is written in C++ and licensed under Apache License v2.0. The tools are compiled on Unix-like systems and Microsoft Windows.

Kaldi was developed based on the demand for an open-source toolkit that deals with finite-state transducer (FST) based framework, and have detailed documentation and scripts for building recognition systems. Some of the other features included in Kaldi are extensive linear algebra support, generic algorithms, complete recipes for building speech recognition systems, and thorough testing [15].

Recipes

The toolkit contains recipes for training acoustic models on various speech corpora such as TIMIT and Wall street journal corpus, including the option to use other speech data. The TIMIT corpus used in this experiment were obtained from Kaldi.

Features

Kaldi also provides feature extraction approaches and waveform-reading code for creating acoustic features and modifying the features. For our experiment, MFCCs and fMLLR features were extracted from the toolkit. The MFCCs were extracted using a 25 ms window and a 10 ms frameshift.

Targets

The targets were available in Kaldi through forced-alignment, where GMM-HMM models had already realigned the transcription at the state level to the speech material. In total, 30 alignment files were obtained in the toolkit. Each alignment file contains utterances, utterance identifiers, and a list of integer identifiers, one for each frame in the utterance. In the toolkit, the identifiers are called transition models (final state transducer). The correspondence between HMM state and the transition models were found by converting the ids into PDF ids. In Kaldi's terminology, the PDFs are HMM states, where there were in total 144 states for the monophones and 1896 states for the triphones.

Phoneme mapping

The state labels mentioned in the previous section had corresponding phonemes. Each monophone provided three emitting states (or PDFs), making in total of 48 phonemes (144 states / 3 = 48). For the triphones, the number of states per phoneme was more unsymmetric as individual phonemes had more states to it than others. The mapping between HMM states and phonemes were found in the Kaldi toolkit.

6.2.2 Python 3

The teacher-student networks were implemented using the deep learning frameworks Tensorflow [1] and Keras [2] in Python 3.7.4. Both the teacher models and the student models were trained on 20 epochs. The performance of the DNNs was monitored over each epoch. See Appendix A, B, C and D to view the Python codes. The codes are repetitive and similar; therefore, Python libraries are not included in Appendix B, C, and D, and dictionaries with the states will not be included in Appendix C and D.

Feature extraction and targets

The feature vectors were normalized with zero mean and a unit variance over the whole training set. These features were then concatenated and fed into the DNN. For the targets, they were one hot encoded and concatenated before training them on the DNNs.

Mapping states to phonemes

The misclassified states were compared to the true labels in each frame. If the misclassified state and true state for the same frame corresponded to the same phoneme, for example, state 4 was classified as state 3, then it will be counted as a correctly classified phoneme because both of those states belong to phoneme /aa/. Afterward, the total amount of times the classes were misclassified but were in the same phoneme was summed up. By summing this number and the number of the total amount of times the model correctly classified states in each frame, the actual correctly classified phone accuracy rate is found. The phone accuracy rate was evaluated on the test set, and was subtracted from 100 to find the phoneme error rate (PER).

Data generator

The triphones were too large and memory consuming. The GPUs have around 11GB of RAM, which was not enough to generate the triphones in Python. Therefore, the data set were processed using a data generator. The Python package Keras provided the framework. Rather than loading the entire data set at once, a data generator generated 12929 data batches to the DNN. The number was chosen because it provided the highest training and validation accuracy for our models. The generator had trained each full epoch on 87 steps of generate batches. The step number was based on the formula

$$\left[\frac{\#samples}{batch\ size}\right], \tag{6.1}$$

with batch size equal to 12929 and a total of 1124823 samples.

6.2.3 Aulus1

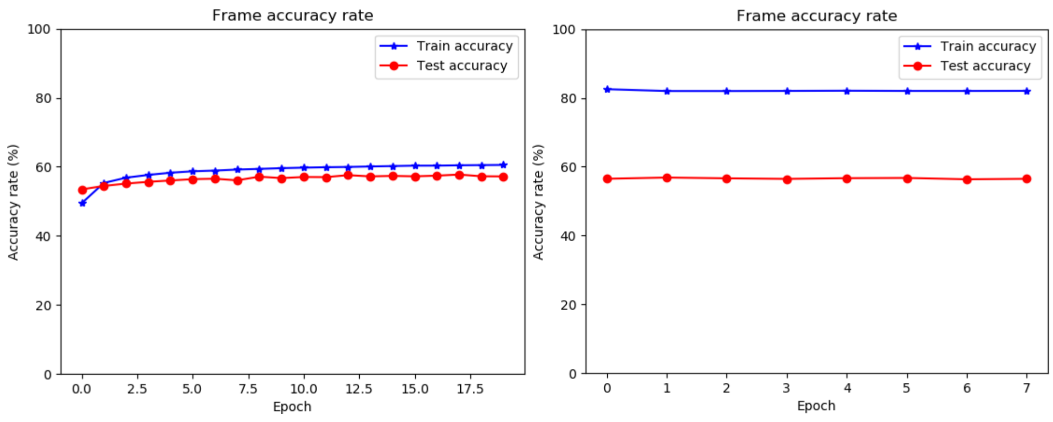
The experiments were run on 64-bit Ubuntu 18.04, using Aulus1 machine at the Norwegian university of science and technology (NTNU). The Aulus1 machine has 16 core Intel(R) Core(TM) i7-5960X CPU @ 3.00GHz processor, with with 32 GB of RAM and 2 x GeForce GTX 1080 Ti GPUs.

Chapter 7

Results

In the following chapter, the experimental activity conducted to assess the proposed teacher/student (T/S) network is described. The experiments reported in the following are based on a DNN frame-by-frame classifier. The speech recognition performance will be reported for the TIMIT corpus.

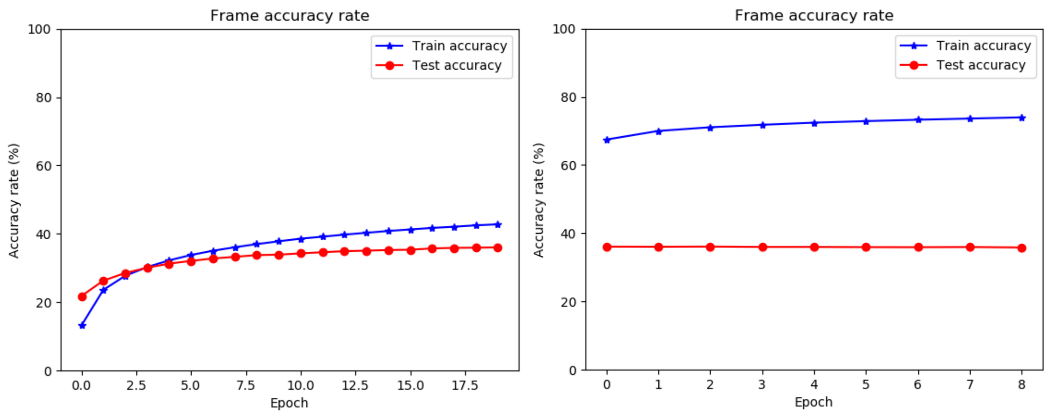
7.1 Frame rate accuracy



(a) Frame accuracy of teacher model. **(b)** Frame accuracy of student model.

Figure 7.1: Frame accuracies of T/S models fed with 13 MFCC features and monophones.

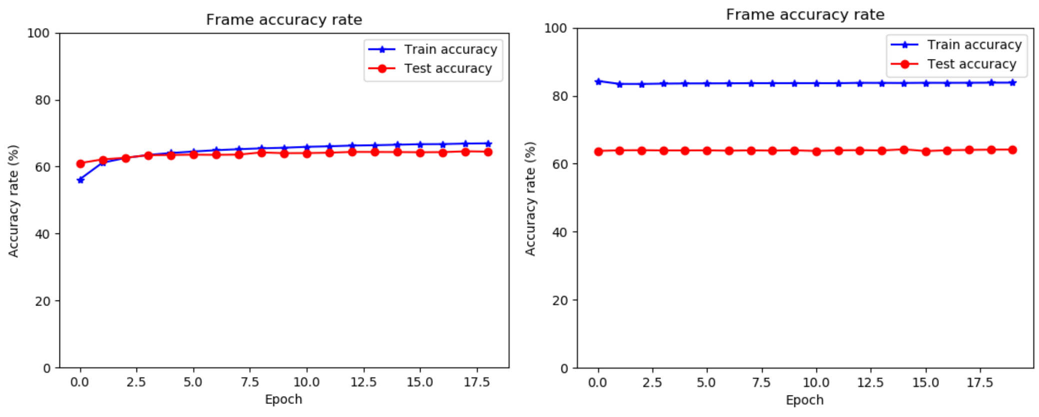
Figure 7.1 shows the frame accuracy rate for the teacher model and the student model, respectively. The teacher model's performance improves for the majority of epochs on both training data and test data. Contrarily, the student models performance is slightly improving on the training data but not on test data. Besides, the student model is overfit by a large margin.



(a) Frame accuracy of teacher model. **(b)** Frame accuracy of student model.

Figure 7.2: Frame accuracies of T/S models fed with 13 MFCC features and triphones.

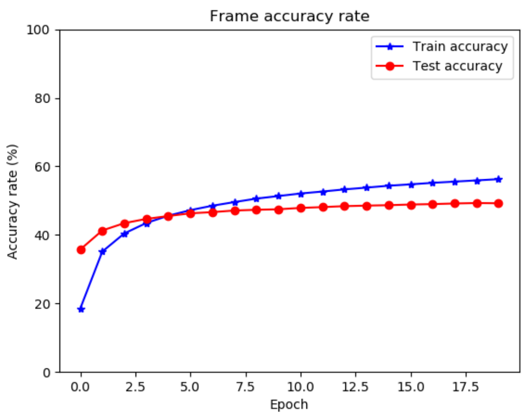
The performance of the teacher model and student model when fed with MFCCs and triphone targets are shown in Figure 7.2. The student model overfits for this case as well.



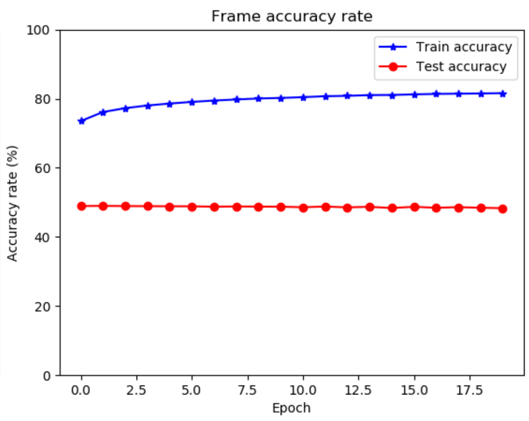
(a) Frame accuracy of teacher model. **(b)** Frame accuracy of student model.

Figure 7.3: Frame accuracies of T/S models fed with fMLLR features and monophones.

In Figure 7.3, the teacher model and student model are trained on fMLLR features and monophone targets. Compared to the teacher model, the student model achieves higher accuracy on the training data, but worse on test data.



(a) Frame accuracy of teacher model.



(b) Frame accuracy of student model.

Figure 7.4: Frame accuracies of T/S models fed with fMLLR features and triphones.

In Figure 7.4, the frame accuracy rates for the T/S models trained on triphone models and fMLLR features are shown.

Table 7.1: Frame accuracy rate (%) on teacher models based on the training set, test set and validation set on the TIMIT database.

Results on TIMIT				
Feature	Phoneme	training acc. (%)	valid. acc. (%)	test acc. (%)
MFCC	CI	60.63	57.47	56.65
MFCC	CD	39.23	34.35	34.91
fMLLR	CI	66.71	64.24	64.21
fMLLR	CD	56.25	48.53	49.18

The results of training the teacher model on a frame-by-frame classifier are shown in Table 7.1. The models trained on the fMLLR have higher frame accuracy rates than the models trained on MFCCs.

Table 7.2: Frame accuracy rate (%) on student models based on the training set, test set and validation set on the TIMIT database.

Results on TIMIT				
Feature	Phoneme	training acc. (%)	valid. acc. (%)	test acc. (%)
MFCC	CI	82.15	57.22	56.52
MFCC	CD	76.21	34.40	35.03
fMLLR	CI	83.44	63.71	63.64
fMLLR	CD	81.42	48.14	48.77

Lastly, the performance accuracy of the student model is summarized in Table 7.2. Training the student model on fMLLR features and CI targets provided the highest accuracy amongst the presented models.

7.2 Mapped phoneme accuracy

Table 7.3: PER for context-dependent and context-independent models.

Results on TIMIT				
T/S model	**Features**	**CI PER (%)**	**CD PER (%)**	**Epochs**
Teacher	MFCC	35.77	40.24	20
Student	MFCC	36.34	39.77	20
Teacher	fMLLR	27.05	28.75	20
Student	fMLLR	27.42	28.99	20

The PER on the CI and CD models are shown in Table 7.3. The PER was calculated by subtracting 100 by the phone accuracy rate that was obtained from having mapped states to phonemes. For the T/S networks, higher phoneme recognition rates were obtained for fMLLRs compared to the MFCCs. Furthermore, models trained on CI phonemes surprisingly gave a higher phoneme recognition rate than CD phonemes. The phoneme accuracy rate is slightly higher for the teacher model compared with its student model.

7.3 Confusion matrices

Confusion matrices were computed based on the correctly classified and misclassified phonemes. Because there were exceedingly many states and phonemes, the performance of the model is displayed through colors rather than numbers. The phonemes that have been predicted the most are shown as stronger green colors, while the lighter the green colors indicate that the classifier did not make predictions on these phonemes as often. A white pixel indicates that the phoneme was not predicted at all. The y-axis has the true labels, and the x-axis has the predicted labels. The correctly classified phonemes are displayed along the diagonal of the figures below.

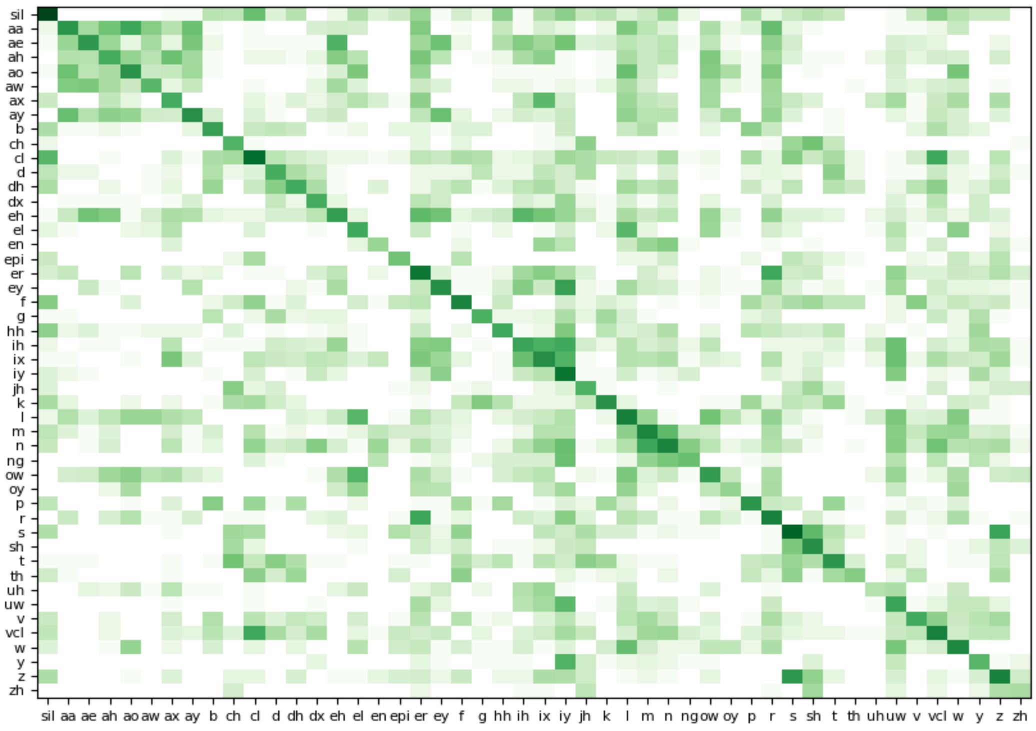


Figure 7.5: CM of CI student model trained on MFCC features.

Figure 7.5 shows the CM for the CI student model trained on MFCC features. The most classified phonemes were around clusters of phonemes {/aa/, /ae/, /ah/, /ao/, /aw/, /ax/, /ay/}, {/cl/, /d/, /dh/}, {/ih/, /ix/, /iy/}, {/l/, /m/, /n/, /ng/} and {/s/, /sh/, /t/}. There are several misclassified phonemes off-diagonal as well.

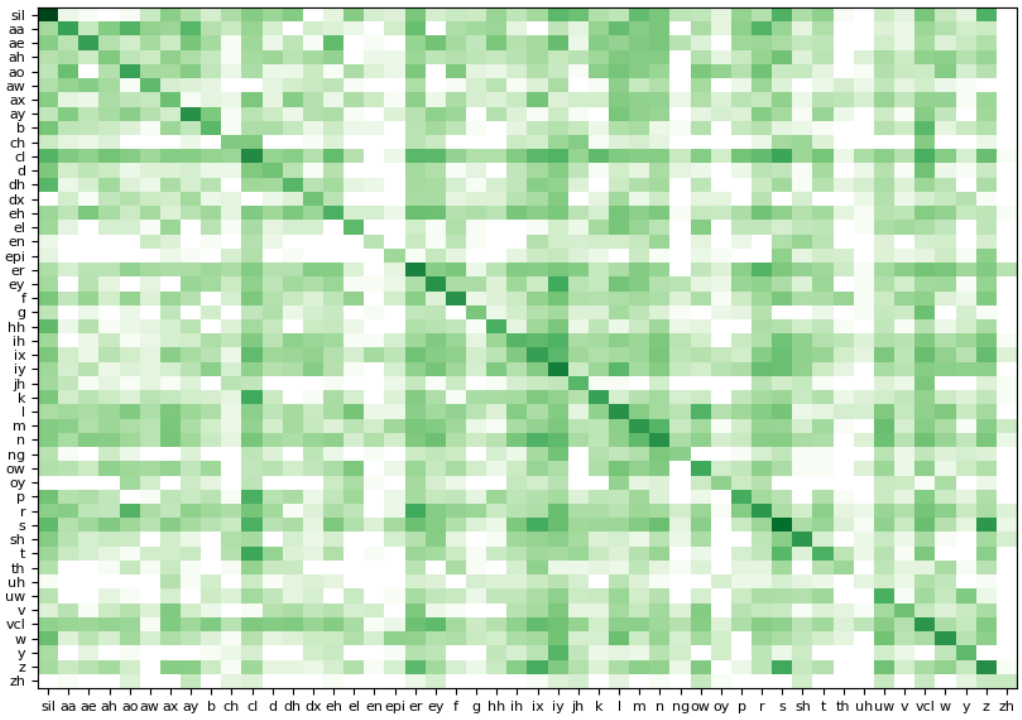


Figure 7.6: CM of CD student model trained on MFCC features.

Figure 7.6 shows the CM for the CD student model trained on MFCC features. The figure shows a significant amount of misclassified phonemes.

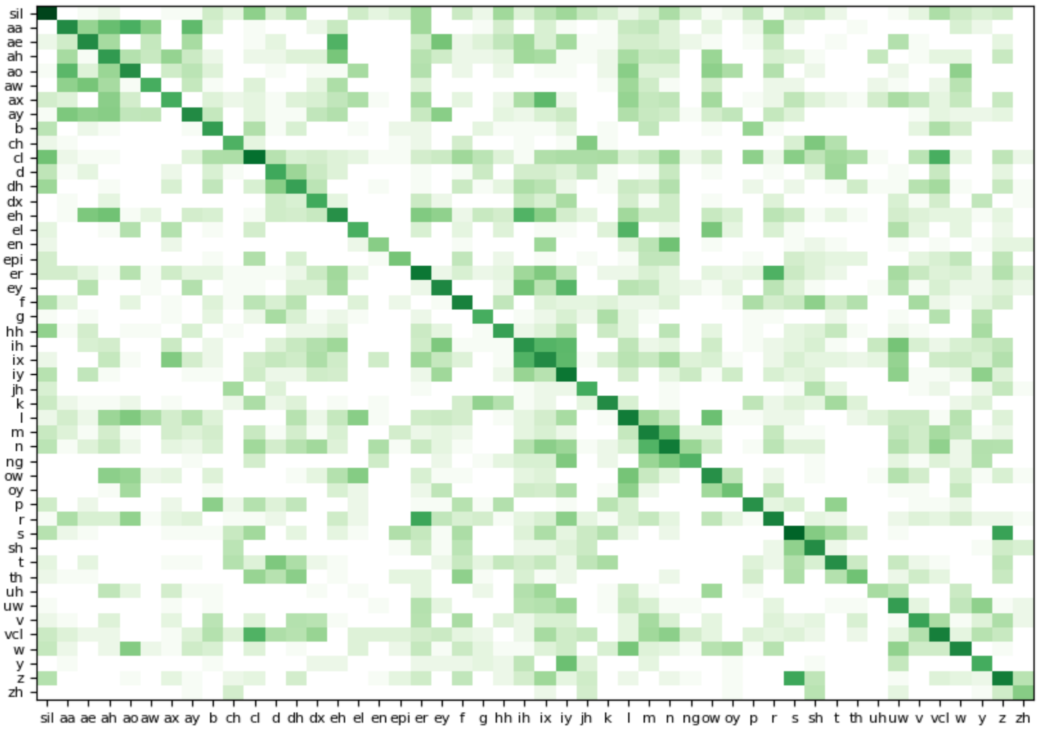


Figure 7.7: CM of CI student model utrained on fMLLR features.

Figure 7.7 shows the CM for the CI student model trained on fMLLR features. The most classified phones were around clusters of phonemes $\{/aa/, /ae/, /ah/, /ao/\}$, $\{/cl/, /d/, /dh/\}$ and $\{/ih/, /ix/, /iy/\}$, $\{/l/, /m/, /n/, /ng/, /ow/\}$.

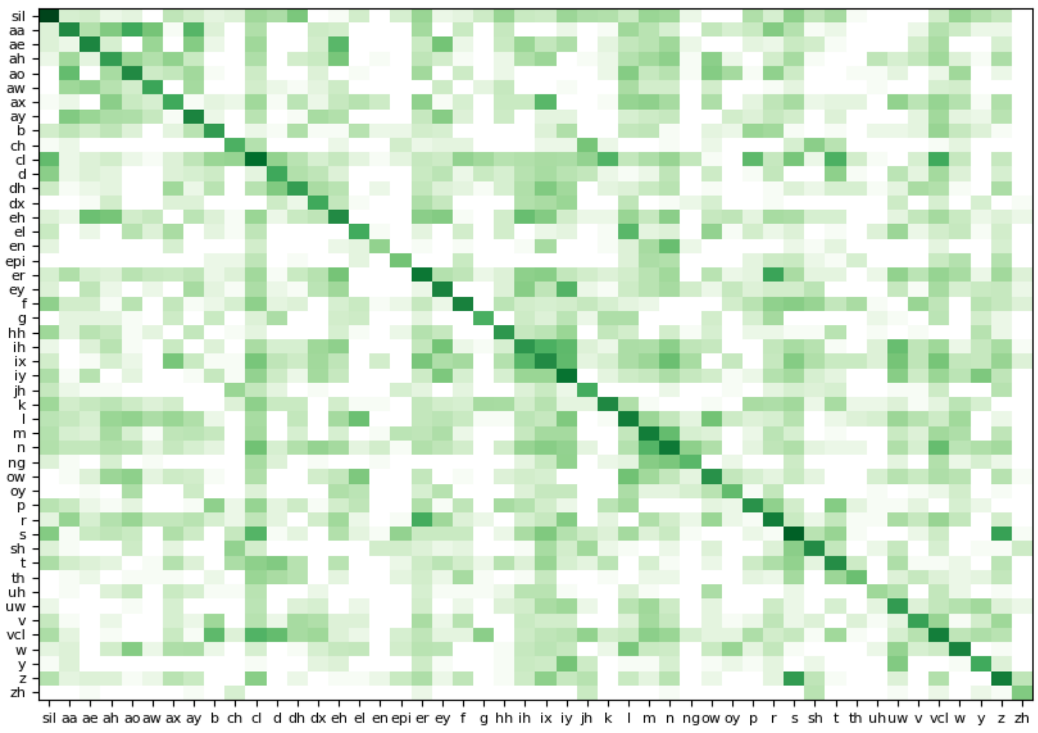


Figure 7.8: CM of CD student model trained on fMLLR features.

Figure 7.8 shows the CM for the CD student model trained on fMLLR features. The most classified phones were along the diagonal and partly on various clusters of phonemes along the figure.

Chapter 8

Discussion

This chapter provides a discussion concerning the results achieved in this work. Section 8.1 starts by discussing the teacher/student (T/S) performance. Section 8.1.1 evaluates the features used in the T/S learning, and Section 8.1.2 evaluates the various phonemes impact on the T/S network. Section 8.1.3 provides an analysis of the confusion matrices. Lastly, Section 8.2 briefly compares our results to state-of-the-art results on T/S learning and explains how one can further extend the work in this thesis.

8.1 Teacher/Student network

The frame accuracy rates in Figures 7.1, 7.2, 7.3 and 7.4 depicts that the teacher models slightly improves for every epoch. The student models, on the other hand, barely improves on unlabeled data. Additionally, Table 7.1 and 7.2 shows that the student model's frame accuracy rates compared to its teacher model are not too indifferent on the validation and test accuracy when trained on MFCC features. For the fMLLR features, the student model had a slightly worse frame accuracy rate on the test set and validation set than its teacher model. These results are rather reasonable, because training on fully labeled data will always outperform a model trained on unlabeled data.

Additionally, the gap between the training accuracy and the test accuracy of the student models is high compared to the gap of teacher models accuracies. This indicates that the student model overfits by a large margin. Because the student model had the same DNN architecture as its teacher model, it can replicate similar accuracy as the teacher model. However, the student model's higher training accuracies are a result of the model starting with a base knowledge gained from the teacher model, and from there improving its knowledge on new input data.

8.1.1 Features

Comparing the confusion matrices in Figure 7.5 to 7.7, one can see that the CM for the MFCC features contains more cases of misclassified phonemes than the CM for the fMLLR features as there are more off-diagonal green colors in Figure 7.5. Likewise with Figure 7.6 compared to Figure 7.8. The results match with the phoneme recognition rates in Table 7.3, and one can conclude that the student model performs better on fMLLR features than for MFCCs. fMLLR features gave a PER of 27.42% for the CI student model, giving a reduced PER rate of 8.92% from the CI student model trained on MFCCs features with PER of 36.34%.

8.1.2 Phoneme models

The frame accuracy rates on the teacher/student models are lower for CD models than the CI models, as shown in Tables 7.1 and 7.2. The lower state accuracy for the CD models could perhaps be a result of some triphones in the given data set having a significant number of states bound to them, compared to other triphones that did not have nearly as many states. Most of the misclassified states among the 1896 labels could, therefore, be between states that belong to the same phonemes. The state accuracy for the monophone models did not differ significantly from the phoneme accuracy rates as each monophone has fewer states (only 3) bound to them.

Context-independent phonemes vs. context-dependent phonemes

Moreover, the triphone model did not outperform the monophone model to classify the most amounts of correct phonemes, which goes against what is mentioned in Chapter 2, Section 2.1.2. The PER was slightly lower for the monophones compared to the triphones. While the PER on monophones was 36.34%, the equivalent PER on triphones was 39.77% for the student model trained with MFCC features, giving a difference of 3.43%. For fMLLR features, the CI PER was 27.42%, and the CD phoneme recognition rate was 28.99% for the student model, giving a difference of 1.57%.

A possible reason for the higher PER on triphone models is that the triphones were fed to the DNN improperly using a data generator. When implementing the data generator in Python, several issues were encountered, including memory issues. Eventually, a training generator was made, where instead of having the generators stop once they had run through the entire training set, the generator would re-run the batch generation process again by starting from the first element of the data sets. Ideally, the generator would have stopped once it had run through the entire training set. However, this was the only way of sending batches of data to the DNNs without encountering any problems. Though not optimal, at least the data was fed to the DNNs.

A last possible explanation, though less likely, for the triphones low accuracy could be that the states of the triphones provided by the Kaldi toolkit are not 100% accurate. The GMM-HMM models may not have given the actual correct states for each frame concerning some cases.

8.1.3 Classified phonemes

Comparing the confusion matrices in Figures 7.5, 7.6, 7.7 and 7.8, the figures for the CI student models have fewer green colors spread around the diagonals than for the CD student models. The CM for the models trained on fMLLRs also has fewer green colors off-diagonal than the models trained on MFCCs, indicating fewer phonemes were misclassified using fMLLR features and monophone targets in our T/S network.

The CD student model trained on MFCCs comes out as the worst model amongst the four mentioned models. This is also confirmed given that the frame accuracy and phoneme recognition rates were the lowest for this specific model.

Moreover, the phonemes that were correctly predicted the most amounts of times were /sil/, /cl/, /er/, /ey/, /f/, /l/, /m/, /n/, /s/, and /z/. These phonemes were often found in the data sets as well. The phonemes that were correctly classified the least amounts of times were /en/, /epi/, /th/, /uh/ and /zh/, possibly because they were not often represented in the data set.

8.2 Future work

There is definitively room for improvements in this work. The work in this thesis was carried out over ten months. Several obstacles occurred during this period, such as program failure and lack of memory. Nevertheless, despite the lack of resources and time limits, our results are still comparison worthy to other research performed on T/S learning.

Our teacher model at its best had a PER of 27.05%, and our student model had a PER of 27.42%. Though not equivalent, the accuracy is not far from the PER of 25.76% from the student baseline presented in the research paper by Li et al. (2017). Compared to the newer research performed by Li et al. (2019), where a PER of 21.3% was achieved, our results are not that impressive. However, though our results are not too impressive, considering that the triphones did not perform optimally in our experiment, it is still a possibility to have even lower PER on our T/S network once the triphones are implemented correctly. Therefore, it would be necessary to fix the data generator or figure out a new way of training a triphone model without encountering any form of memory issues.

Additionally, it would be interesting to experiment with other toolkits as well. Perhaps using the Kaldi toolkit with the Pytorch framework in the Python language, as proposed by Ravanelly et al. (2019), our T/S network could have achieved lower PER on the teacher model and consequently on the student model as well.

Out of personal interest, implementing T/S network on other acoustic models (AMs), such as recurrent neural networks (RNNs) or convolutional neural networks (CNNs), or train our data on an HMM, and comparing those AM to our T/S network trained on DNNs would be intriguing. Besides, our teacher-student networks is just one of many possible ways of implementing semi-supervised learning. It would also have been interesting to try other semi-supervised learning techniques to see how robust our method is.

It is also worth mentioning that the teacher model was initially supposed to be trained on a partly labeled training set, but was trained on a fully labeled training set in this work. Therefore, it is of high interest to see how the teacher model and student model would perform when the teacher model is trained on partly labeled training data and the student on completely unlabeled data. The main goal of semi-supervised learning is to implement a system that gives low PER when trained on partly labeled data due to the lack of annotated data in the real world. Therefore, would a student model outperform a teacher model trained on 30% labeled data?

Chapter 9

Conclusion

Two teacher/student (T/S) networks were implemented to explore semi-supervised learning for ASR. The teacher model was built on a deep neural network (DNN), and the student model was built on the same DNN architecture. The T/S networks were trained on Mel-frequency cepstral coefficients (MFCC) and feature-space maximum likelihood linear regression (fMLLR) features. Both monophones and triphones were used as targets in two separate networks. The features and the targets were obtained through the Kaldi toolkit.

The teacher model was trained on a full set of manually annotated data and a student model trained on a full set of automatically annotated data provided by the teacher model. The models achieved the most optimal results when trained on fMLLR features. A PER of 27.42 % was achieved from the baseline student model when trained on fMLLRs and monophone targets. The worst student model achieved a PER of 39.77% when trained on MFCCs and triphone targets. Additionally, the monophone models outperformed the triphone models. Due to memory issues and inaccurate implementation of data generators for the triphones, the triphones achieved surprisingly lower accuracy rates than expected.

Regarding the T/S network performance, the student model achieved similar accuracies as its teacher model on the test set. However, the student model did not improve its performance on new, unlabeled data. The student models relied heavily on the performance of the teacher models. Hence, if the frame accuracy rates of the teacher model were to be improved, the student model's performance would also improve, considering it copies teacher models behavior.

Although the T/S network for semi-supervised learning shows promising results, our results also indicate that higher accuracy rates are achievable for T/S models if the triphones had been appropriately implemented.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] F. Chollet et al. Keras. `https://keras.io`, 2015.

[3] L. D. Consortium et al. Timit acoustic-phonetic continuous speech corpus. *URL http://www. ldc. upenn. edu/Catalog/CatalogEntry. jsp*, 1993.

[4] M. J. Gales. Maximum likelihood linear transformations for hmm-based speech recognition. *Computer speech & language*, 12(2):75–98, 1998.

[5] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, N. L. Dahlgren, and V. Zue. Timit acoustic-phonetic continuous speech corpus (1990). *URL http://www. ldc. upenn. edu/Catalog/CatalogEntry. jsp.*

[6] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[7] X. Huang, A. Acero, H.-W. Hon, and R. Foreword By-Reddy. *Spoken language processing: A guide to theory, algorithm, and system development.* Prentice hall PTR, 2001.

[8] D. Jurafsky and J. H. Martin. Speech and language processing: An introduction to speech recognition, computational linguistics and natural language processing. *Upper Saddle River, NJ: Prentice Hall*, 2008.

[9] S. Kim, M. L. Seltzer, J. Li, and R. Zhao. Improved training for online end-to-end speech recognition systems. *arXiv preprint arXiv:1711.02212*, 2017.

[10] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[11] Q. Li, C. Zhang, F. Kreyssig, and P. Woodland. Experimental studies on teacher-student training of deep neural network acoustic models.

[12] W. Li, S. Wang, M. Lei, S. M. Siniscalchi, and C.-H. Lee. Improving audio-visual speech recognition performance with cross-modal student-teacher training. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6560–6564. IEEE, 2019.

[13] J. Lyons. Mel frequency cepstral coefficient (mfcc) tutorial. *Practical Cryptography*, 2015.

[14] A. L. Maas, P. Qi, Z. Xie, A. Y. Hannun, C. T. Lengerich, D. Jurafsky, and A. Y. Ng. Building dnn acoustic models for large vocabulary speech recognition. *Computer Speech & Language*, 41:195–213, 2017.

[15] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely. The kaldi speech recognition toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, Dec. 2011. IEEE Catalog No.: CFP11SRW-USB.

[16] R. Price, K.-i. Iso, and K. Shinoda. Wise teachers train better dnn acoustic models. *EURASIP Journal on Audio, Speech, and Music Processing*, 2016(1):10, 2016.

[17] M. Ravanelli, T. Parcollet, and Y. Bengio. The pytorch-kaldi speech recognition toolkit. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6465–6469. IEEE, 2019.

[18] L. Tóth. Phone recognition with hierarchical convolutional deep maxout networks. *EURASIP Journal on Audio, Speech, and Music Processing*, 2015(1):1–13, 2015.

[19] D. Yu and L. Deng. *Automatic speech recognition.* Springer, 2016.

A MFCC and monophones

```
from keras.preprocessing.sequence import TimeseriesGenerator
from keras.models import Sequential
from keras.layers import Input, Dense, Activation, Dropout
from keras.callbacks import TensorBoard, ModelCheckpoint, EarlyStopping
from keras.callbacks import Callback
from keras.utils import to_categorical
from keras.models import Model, load_model
from sklearn.metrics import classification_report, confusion_matrix
from matplotlib.ticker import PercentFormatter
import matplotlib.ticker as mtick

from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier

import matplotlib.pyplot as plt
import tensorflow as tf
import tensorflow.keras as keras
import numpy as np
import pickle
import pandas as pd
import random
import sys
import itertools

#Show full numpy array
np.set_printoptions(threshold=sys.maxsize)

#We have the saved pickle file, now we need to access the pickled file:
# open a file, where you stored the pickled data
file = open('mfcc_train.pckl', 'rb')

# dump information to that file
mfcc_train = pickle.load(file)
# close the file
file.close()
```

```
file = open('mfcc_test.pckl', 'rb')
mfcc_test = pickle.load(file)
file.close()

file = open('mfcc_dev.pckl', 'rb')
mfcc_val = pickle.load(file)
file.close()

#Monophones
#Targets for the training data:
targets = {}
nTargets = 0
for n in range(1, 31):
    with open('mono_ali.'+str(n)+'.pdf.txt') as f:
        monoali1 = [x.strip() for x in f.readlines()]
    for item in monoali1:
        #print(item)
        data = item.split() #Split a string into a list where each word is a
            list item
        #print(data)
        numdata = np.array([int(el) for el in data[1:]]) #here each words
            become a item
        #print(numdata)
        targets[data[0]] = numdata
        nTargets = np.max([nTargets, numdata.max()])

nTargets += 1


devTargets = {}
dev_nTargets = 0

for n in range(1,2):
    #Targets for the validation data:
    with open('mono_ali_dev.'+str(n)+'.pdf.txt') as f:
        monoali_dev = [x.strip() for x in f.readlines()]
    for item in monoali_dev:
        data_dev = item.split() #Split a string into a list where each word is
            a list item
        numdata_dev = np.array([int(el) for el in data_dev[1:]]) #here each
            words become a item
        devTargets[data_dev[0]] = numdata_dev
        dev_nTargets = np.max([dev_nTargets, numdata_dev.max()])

dev_nTargets += 1
```

```
testTargets = {}
test_nTargets = 0
for n in range(1,2):
    with open('mono_ali_test.'+str(n)+'.pdf.txt') as f:
        monoali_test = [x.strip() for x in f.readlines()]
    for item in monoali_test:
        data_test = item.split() #Split a string into a list where each word is
            a list item
        numdata_test = np.array([int(el) for el in data_test [1:]])  #here each
            words become a item
        testTargets[data_test [0]]  = numdata_test
        test_nTargets = np.max([test_nTargets, numdata_test.max()])
test_nTargets += 1


phonemes = ['sil', 'aa', 'ae', 'ah', 'ao', 'aw', 'ax', 'ay', 'b', 'ch', 'cl', '
    d', 'dh', 'dx', 'eh', 'el', 'en', 'epi', 'er', 'ey', 'f', 'g', 'hh', 'ih',
    'ix', 'iy', 'jh', 'k', 'l', 'm', 'n', 'ng', 'ow', 'oy', 'p', 'r', 's', 'sh'
    , 't', 'th', 'uh', 'uw', 'v', 'vcl', 'w', 'y', 'z', 'zh']


states = {'sil': [0, 1, 2], 'aa': [3, 4, 5], 'ae': [6, 7, 8], 'ah': [9, 10,
    11], 'ao': [12, 13, 14], 'aw': [15, 16, 17],  'ax': [18, 19, 20], 'ay':
    [21, 22, 23], 'b': [24, 25, 26], 'ch': [27, 28, 29], 'cl': [30, 31, 32], '
    d': [33, 34, 35], 'dh': [36, 37, 38], 'dx': [39, 40, 41], 'eh': [42, 43,
    44], 'el': [45, 46, 47], 'en': [48, 49, 50], 'epi': [51, 52, 53], 'er':
    [54, 55, 56], 'ey': [57, 58, 59], 'f': [60, 61, 62], 'g': [63, 64, 65], '
    hh': [66, 67, 68], 'ih': [69, 70, 71], 'ix': [72, 73, 74], 'iy': [75, 76,
    77], 'jh': [78, 79, 80], 'k': [81, 82, 83], 'l': [84, 85, 86], 'm': [87,
    88, 89], 'n': [90, 91, 92], 'ng': [93, 94, 95], 'ow': [96, 97, 98], 'oy':
    [99, 100, 101], 'p': [102, 103, 104], 'r': [105, 106, 107], 's': [108, 109,
     110], 'sh': [111, 112, 113], 't': [114, 115, 116], 'th': [117, 118, 119],
    'uh': [120, 121, 122], 'uw': [123, 124, 125], 'v': [126, 127, 128], 'vcl':
    [129, 130, 131], 'w': [132, 133, 134], 'y': [135, 136, 137], 'z': [138,
    139, 140], 'zh': [141, 142, 143]}

def frameConcat(x,splice, splType):
    validFrm = int( np.sum(np.sign( np.sum( np.abs(x), axis=1) )) )
    nFrame, nDim = x.shape

    if ( splType == 1):
        spl = splice
        splVec = np.arange(0, int(2*spl+1), 1)
    else:
        spl  = int(2*splice)
        splVec = np.arange(0, int(2*spl+1), 2)

    xZerosPad = np.vstack([np.zeros((spl, nDim)), x[0:validFrm ,:],  np.zeros((
         spl,  nDim))])
    xConc = np.zeros( (validFrm, int(nDim*(2*splice+1))) )

    for iFrm in range(validFrm):
        xConcTmp = np.reshape(xZerosPad[iFrm+splVec,:], (1,int((2*splice+1)*
             nDim)) )
        xConc[iFrm, :]  = xConcTmp
    return xConc

model = Sequential()
model.add(Dense(143, activation='relu', input_shape=(143,)))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(256, activation='relu'))
model.add(Dense(nTargets, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['
     accuracy'])
model.summary()


#13 MFCC:
x_train  = np.zeros((0,  143)) #13*11, 5 frames on each side of the current  mfcc
x_test  = np.zeros((0,  143))
x_val  = np.zeros((0,  143))
y_train  = np.zeros((0,  nTargets))
y_test  = np.zeros((0,  test_nTargets))
y_val  = np.zeros((0,  dev_nTargets))


#Monophones:
for keys in mfcc_train.keys():
    mfccarray = mfcc_train[keys]
    x_mean = np.mean(mfccarray, axis=0)
    x_std  = np.std(mfccarray, axis=0)
    mfcctrain_normalized = ( mfccarray − x_mean ) / x_std
    trainConc=frameConcat(mfcctrain_normalized, 5, 1) #should give 13*11
    x_train  = np.vstack((x_train,  trainConc)) #concatenate mfcc

    targetsarray  = targets[keys]
    Labels = np.eye(nTargets)
    targetOneHot = Labels[targetsarray, :]
    y_train = np.vstack((y_train, targetOneHot)) #concatenated targets


for keys in mfcc_val.keys():
    valarray = mfcc_val[keys]
    mfccval_normalized = ( valarray - x_mean ) / x_std
    valConc=frameConcat(mfccval_normalized, 5, 1)
    x_val = np.vstack((x_val, valConc))

    targetsarray_val = devTargets[keys]
    Labels_val = np.eye(dev_nTargets)
    targetOneHot_val = Labels_val[targetsarray_val, :]
    y_val = np.vstack((y_val, targetOneHot_val))


for keys in mfcc_test.keys():
    testarray = mfcc_test[keys]
    mfcctest_normalized = ( testarray - x_mean ) / x_std
    testConc=frameConcat(mfcctest_normalized, 5, 1)
    x_test = np.vstack((x_test, testConc))

    targetsarray_test = testTargets[keys]
    Labels_test = np.eye(test_nTargets)
    targetOneHot_test = Labels_test[targetsarray_test, :]
    y_test = np.vstack((y_test, targetOneHot_test))



callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=6)
history = model.fit(x_train, y_train, validation_data=(x_val, y_val),
    batch_size=256, epochs=20, callbacks=[callback], verbose=1, shuffle=True)
numberOfEpochs = len(history.history['loss'])


#Test model: (on full data set)
score, acc = model.evaluate(x_test, y_test, batch_size=256, verbose=1)
print('Test score:', score)
print('Test accuracy:', acc)

targetClass = np.where(y_test==1)[1]
predictedClass = model.predict_classes(x_test)



#MAKES LIST OF LIST FROM DICTIONARY
```

```
statesValues = []
for keys in states.keys():
    phonemeStates = states[keys] #this is a  list
    statesValues.append(phonemeStates)


### PHONEME RECOGNITION ####

mapedState = [0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 7,
     7, 7, 8, 8, 8, 9, 9, 9, 10, 10, 10, 11, 11, 11, 12, 12, 12, 13, 13, 13,
    14, 14, 14, 15, 15, 15, 16, 16, 16, 17, 17, 17, 18, 18, 18, 19, 19, 19, 20,
     20, 20, 21, 21, 21, 22, 22, 22, 23, 23, 23, 24, 24, 24, 25, 25, 25, 26,
    26, 26, 27, 27, 27, 28, 28, 28, 29, 29, 29, 30, 30, 30, 31, 31, 31, 32, 32,
     32, 33, 33, 33, 34, 34, 34, 35, 35, 35, 36, 36, 36, 37, 37, 37, 38, 38,
    38, 39, 39, 39, 40, 40, 40, 41, 41, 41, 42, 42, 42, 43, 43, 43, 44, 44, 44,
     45, 45, 45, 46, 46, 46, 47, 47, 47]

# I've made two ways of checking whether the predicted states belong to the
    same phoneme as the target states, both methods are correct:

#Method 1:
counter = 0
for i in range(len(predictedClass)):
    if predictedClass[i] != targetClass[i]:
        if mapedState[predictedClass[i]] == mapedState[targetClass[i]]:
            counter += 1

'''
#Method 2:
correctPhonemePosition = []
stateInPhoneme = []
for i in range(len(y_test)):
    if (predictedClass[i] != targetClass[i]):
        for index, nested_list in enumerate(newArray):
            if predictedClass[i] in nested_list and targetClass[i] in
                nested_list :
                correctPhonemePosition.append(index)
                stateInPhoneme.append(i)

'''

###Recalculating accuracy
correct = 0
for j in range(len(y_test)):
    if predictedClass[j] == targetClass[j]:
        correct +=1


correctPhonemes = counter+correct

#Phoneme recognition:
newAccuracy = 100 * (correctPhonemes/len(x_test))
print('Phoneme recognition accuracy: ', newAccuracy)

### END PHONEME RECOGNITION ###



### SSL: Student model###

predicted = model.predict(x_train, batch_size=256, verbose=1)

model.save('model1.h5')
del model


student = load_model('model1.h5')
student.summary()


callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=6)
history1 = student.fit(x=x_train, y=predicted, validation_data=(x_val, y_val),
     batch_size=256, epochs=20, callbacks=[callback], verbose=1, shuffle=True)

score, acc = student.evaluate(x_test, y_test)
print('Test score: ', score)
print('Test accuracy: ', acc)


### PHONEME RECOGNITION ###
predictedClass = student.predict_classes(x_test)

stateInPhoneme = []
for i in range(len(y_test)):
    if (predictedClass[i] != targetClass[i]):
        for index, nested_list in enumerate(statesValues):
            if predictedClass[i] in nested_list and targetClass[i] in
                 nested_list :
                stateInPhoneme.append(i)
                #print(index)
```

```
#Recalculating accuracy
correct  = 0
for j in range(len(y_test)):
    if predictedClass[j] == targetClass[j]:
        correct +=1
print('correct: ', correct)


correctPhonemes = len(stateInPhoneme)+correct)

newAccuracy = 100 * (correctPhonemes/len(predictedClass))
print('Phoneme recognition accuracy: ', newAccuracy)

### END PHONEME RECOGNITION ###

posPhoneme = []
posTarget = []
for i in range(len(y_test)):
        for index, nested_list in enumerate(statesValues):
            if predictedClass[i] in nested_list :
                posPhoneme.append(index)
            if targetClass[i] in nested_list :
                posTarget.append(index)

y_pred = posPhoneme
y = posTarget


cm=confusion_matrix(y, y_pred)

cm_df2 = np.log(cm_df)
plt.figure(figsize =(31, 22))
plt.imshow(cm_df2, interpolation='nearest', aspect='auto', cmap='Greens')
tick_marks = np.arange(len(phonemes))
plt.xticks(tick_marks, phonemes, fontsize=17)
plt.yticks(tick_marks, phonemes, fontsize=17)
plt.savefig('CMmonomfcc.png')
```

B MFCC and triphones

```
np.set_printoptions(threshold=sys.maxsize)


file = open('mfcc_train.pckl', 'rb')

# dump information to that file
mfcc_train = pickle.load(file)
# close the file
file.close()

file = open('mfcc_test.pckl', 'rb')
mfcc_test = pickle.load(file)
file.close()

file = open('mfcc_dev.pckl', 'rb')
mfcc_val = pickle.load(file)
file.close()

states = {'sil': [0, 54, 49], 'aa': [1, 134, 234, 394, 426, 574, 592, 610, 624,
     816, 817, 849, 1065, 1213, 1315, 1470, 1506, 1507, 1595, 1596, 1618, 1628,
     1666, 1691, 1721, 57, 115, 283, 380, 624, 791, 903, 938, 941, 1008, 1083,
    1131, 1359, 1400, 1505, 1573, 95, 165, 321, 387, 541, 682, 699, 749, 1414,
    1597, 1784, 1803], 'ae': [59, 168, 296, 331, 383, 479, 606, 769, 806, 973,
    976, 1201, 1335, 1426, 1456, 1737, 1800, 1830, 52, 666, 784, 841, 922,
    1095, 1119, 1298, 1306, 1347, 1406, 1424, 1471, 1517, 1594, 1648, 1670,
    1676, 1678, 1717, 1797, 2, 152, 198, 215, 464, 505, 716, 892, 911, 970,
    1021, 1023, 1192, 1281], 'ah': [3, 143, 245, 585, 586, 884, 1040, 1154,
    1211, 1288, 1316, 1459, 1532, 1539, 1571, 1779, 1782, 1815, 1826, 1865, 77,
     260, 452, 570, 751, 821, 885, 1117, 1343, 1550, 1570, 1713, 1729, 1852,
    1856, 166, 356, 581, 727, 851, 989, 1006, 1078, 1257, 1713, 1714], 'ao':
    [4, 69, 157, 225,420, 485, 523, 601, 602, 622, 638, 665, 906, 936, 1019,
    1068, 1262, 1362, 1443, 1465, 1525, 1582, 1635, 1642, 1707, 1791, 1792,
    167, 444, 781, 889, 1019, 1224, 1250, 1536, 1566, 1790, 100, 117, 186, 219,
     358, 623, 652, 765, 824, 1546, 1754, 1789], 'aw': [612, 628, 636, 811,
    1181,1331, 1616, 1814, 76, 521, 932, 1003, 1232, 1549, 5, 110, 270, 330,
    812, 994, 1175, 1294, 1428, 1831, 1843],  'ax': [112, 206, 466, 562, 876,
```

951, 999, 1203, 1333, 1436, 1547, 1654, 1719, 1723, 1734, 1785, 1861, 1869,
6, 177, 218, 278, 291, 323, 425, 549, 683, 750, 804, 1070, 1196, 1233,
1252, 1348, 1806, 1810, 1861, 1885, 291, 336, 466, 549, 561, 654, 750, 753,
804, 860, 1096, 1097, 1144, 1233, 1364, 1464, 1527, 1806, 1817, 1854,
1880], 'ay': [7, 75, 326, 441, 609, 772, 773, 913, 955, 1012, 1035, 1243,
1275, 1313, 1346, 1444, 1542, 1560, 1602, 1606, 1643, 1644, 1645, 217, 857,
909, 1012, 1164, 1317, 1318, 1329, 1346, 1629, 1639, 50, 92, 181, 339,
355, 375, 378, 544, 546, 745, 833, 1141, 1165, 1208, 1329, 1346, 1447,
1492, 1553], 'b': [481, 626, 757, 802, 943, 1177, 1278, 98, 391, 402, 589,
801, 871, 943, 944, 1277, 8, 158, 265, 415, 584, 715, 748, 912, 943, 1066],
'ch': [9, 135, 1029, 1475, 1520, 1775, 197, 332, 545, 759, 958, 1029,
1220, 1412, 1610, 1674], 'cl': [10, 60, 179, 244, 248, 320, 351, 366, 418,
448, 467, 511, 551, 583, 607, 639, 879, 887, 977, 1045, 1058, 1079, 1110,
1182, 1189, 1319, 1388, 1395, 1423, 1457, 1460, 1544, 1556, 1637, 1682,
1686, 1733, 88, 170, 220, 250, 251, 294, 338, 365, 595, 598, 831, 858, 861,
878, 916, 931, 1004, 1044, 1088, 1176, 1241, 1370, 1373, 1386, 1515, 1524,
1540, 1608, 1636, 1673, 1684, 1686, 1701, 1736, 1751, 56, 129, 146, 169,
211, 243, 293, 308, 386, 408, 411, 414, 721, 831, 868, 1024, 1087, 1174,
1349, 1425, 1486, 1564, 1683, 1700, 1703, 1766], 'd': [11, 498, 502, 508,
1009, 1401, 1600, 1726, 205, 502, 554, 898, 918, 1325, 1420, 1452, 126,
184, 205, 501, 593, 762, 893, 894, 920, 1325, 1387], 'dh': [12, 125, 335,
518, 1067, 1129, 1149, 1296, 1304, 1320, 196, 359, 635, 760, 946, 1848,
1855, 108, 368, 785, 908, 1129, 1194, 1273, 1599, 1794, 1809], 'dx': [651,
778, 832, 1013, 1240, 1502, 1662, 149, 1114, 1476, 1702, 13, 288, 633,
1094, 1310, 1503, 1529], 'eh': [14, 164, 317, 403, 416, 461, 486, 573, 670,
741, 901, 1011, 1059, 1237, 1270, 1378, 1498, 1584, 1601, 1652, 1658,
1663, 1689, 1748, 1767, 1801, 1874, 1876, 63, 102, 162, 423, 462, 531, 687,
901, 1120, 1230, 1251, 1260, 1495, 1706, 1812, 136, 202, 207, 377, 407,
492, 631, 708, 910, 1121, 1225, 1258, 1495, 1561], 'el': [15, 547, 648,
1055, 1402, 1581, 1888, 1893, 104, 455, 718, 1461, 1485, 1842, 1879, 173,
273, 342, 815, 995, 1336, 1461, 1866, 1878], 'en': [16, 771, 983, 1763,
1894, 290, 1091, 1323, 1339, 1895, 228, 525, 657, 1053, 1836], 'epi': [17,
345, 924, 1687, 1732, 345, 364, 528, 872, 1718, 1732, 872, 877, 963, 1338,
1718, 1732, 1802], 'er': [18, 124, 472, 520, 522, 538, 577, 616, 681, 722,
810, 827, 882, 921, 1071, 1173, 1183, 1212, 1434, 1451, 1453, 1462, 1513,
1563, 1578, 1580, 1603, 1604, 1656, 1657, 1669, 51, 187, 192, 249, 341,
412, 480, 490, 565, 605, 770, 865, 888, 917, 1038, 1046, 1199, 1231, 1383,
1391, 1516, 1534, 1559, 1638, 1655, 1697, 74, 127, 145, 292, 314, 406, 491,
537, 559, 600, 742, 803, 862, 919, 925, 975, 1005, 1060, 1116, 1158, 1185,
1467, 1512, 1649], 'ey': [19, 147, 174, 333, 346, 433, 504, 823, 1010,
1082, 1268, 1301, 1321, 1382, 1523, 1626, 1795, 1805,1839, 1841, 1882, 71,
240, 445, 625, 1089, 1153, 1299, 1398, 1545, 1557, 1585, 1849, 1857, 1883,
1886, 1889, 1892, 55, 212, 247, 252, 434, 517, 603, 676, 705, 867, 899,
935, 1113, 1269, 1667, 1681, 1783, 1891], 'f': [20, 266, 376, 470, 558,
560, 863, 956, 980, 1187, 1358, 1407, 68, 107, 216, 347, 552, 557, 572,

674, 743, 1168, 254, 347, 515, 572, 744, 754, 934, 1167, 1186, 1198, 1256],
'g': [21, 474, 1478, 1479, 257, 318, 685, 907, 1169, 161, 257, 720, 826,
1477, 1572], 'hh': [396, 580, 655, 1105, 1133, 1350, 1354, 1396, 1634,
1664, 1675, 1679, 1828, 1829, 148, 155, 350, 566, 755, 880, 1289, 1538,
1625, 1832, 22, 286, 316, 859, 968, 971, 1161, 1172, 1450, 1497, 1765], 'ih
': [132, 153, 263, 389, 487, 510, 691, 818, 883, 896, 959, 1034, 1061,
1126, 1244, 1380, 1390, 1484, 1494, 1499, 1537, 1609, 1617, 1646, 1647,
1709, 1730, 1770, 1845, 1859, 1860, 1871, 1875, 23, 62, 185, 284, 468, 478,
702, 792, 839, 967, 1041, 1309, 1409, 1469, 1659, 1693, 1735, 87, 180,
194, 325, 329, 582, 619, 627, 647, 768, 828, 967, 1085, 1586, 1735], 'ix':
[24, 111, 235, 334, 354, 410, 442, 482, 599, 663, 700, 1150, 1267, 1324,
1371, 1558, 1612, 1641, 1712, 1725, 1727, 1771, 1781, 1808, 1824, 1825,
1837, 1840, 1870, 81, 89, 259, 281, 392, 393, 454, 659, 830, 1049, 1104,
1200, 1253, 1291, 1326, 1330, 1463, 1680, 1710, 1769, 1808, 1820, 1840,
1850, 1872, 116, 222, 281, 322, 370, 453, 550, 926, 1049, 1052, 1115, 1276,
1291, 1381, 1411, 1458, 1631, 1680, 1722, 1728, 1823], 'iy': [70, 191,
233, 303, 306, 357, 449, 512, 563, 618, 634, 783, 842, 869, 964, 996, 1190,
1202, 1218, 1254, 1342, 1369, 1431, 1432, 1437, 1449, 1472, 1493, 1543,
1757, 1758, 1772, 1774, 25, 123, 236, 237, 542, 621, 667, 713, 761, 965,
1007, 1138, 1242, 1284, 1356, 1393, 1521, 1551, 1552, 1633, 1653, 1661,
1704, 1756, 53, 103, 119, 189, 213, 241, 421, 422, 424, 646, 814, 844, 900,
942, 1015, 1036, 1074, 1122, 1138, 1145, 1229, 1368, 1372, 1551, 1715,
1716], 'jh': [26, 1226, 1367, 1755, 209, 253, 1018, 144, 344, 604, 678,
680, 1421, 1508, 1838], 'k': [27, 246, 315, 495, 794, 1016, 1101, 1227,
1340, 1341, 1468, 114, 279, 477, 579, 693, 840, 915, 1102, 1147, 1209,
1214, 1312, 1496, 79, 203, 255, 277, 328, 381, 430, 432, 519, 649, 686,
738, 793, 822, 969, 1293, 1389, 1496], 'l': [28, 67, 99, 276, 343, 382,
514, 540, 594, 669, 746, 766, 800, 836, 886, 949, 1048, 1072, 1075, 1081,
1178, 1179, 1180, 1188, 1271, 1290, 1344, 1394, 1510, 1569, 86, 122, 182,
287, 469, 553, 696, 735, 736, 789, 864, 1028, 1106, 1108, 1109, 1283, 1305,
1455, 1568, 1630, 139, 188, 227, 340, 367, 379, 597, 664, 735, 805, 837,
940, 984, 1076, 1090, 1107, 1140, 1142, 1204, 1223, 1345, 1510, 1531], 'm':
[29, 101, 299, 447, 656, 672, 689, 697, 954, 960, 978, 1063, 1124, 1125,
1143, 1156, 1157, 1577, 160, 443, 483, 734, 854, 945, 978, 1025, 1063,
1084, 1086, 1171, 1473, 1535, 1567, 83, 91, 138, 256, 274, 275, 483, 484,
587, 608, 673, 731, 838, 937, 1026, 1155, 1374, 1504], 'n': [30, 66, 567,
611, 617, 694, 853, 891, 979, 1054, 1162, 1195, 1249, 1376, 1441, 1446,
1575, 1579, 1607, 1627, 1692, 1743, 1760, 1762, 105, 193, 232, 319, 360,
398, 526, 588, 752, 756, 852, 1002, 1152, 1195, 1248, 1249, 1274, 1332,
1375, 1377, 1518, 82, 106, 172, 175, 352, 372, 397, 429, 588, 591, 630,
643, 834, 835, 875, 985, 1032, 1092, 1100, 1205, 1363,1500, 1695, 1745,
1773], 'ng': [690, 717, 902, 1351, 1385, 1665, 137, 413, 719, 966, 1014,
1206, 31, 280, 300, 571, 905, 1352, 1385, 1593, 1708], 'ow': [64, 109, 363,
494, 578, 613, 662, 763, 1043, 1247, 1279, 1541, 1583, 1632, 1677, 1711,
1819, 1847, 1877, 178, 262, 458, 726, 847, 991, 1062, 1287, 1300, 1548,

```
1555, 1696, 1834, 32, 231, 261, 298, 698, 737, 992, 1042, 1132, 1308, 1487,
 1746, 1787, 1833, 1835], 'oy': [576, 895, 1448, 1483, 1761, 130, 1148,
1864, 33, 190, 950, 1307, 1430, 1739], 'p': [34, 615, 645, 764, 796, 1454,
72, 369, 404, 417, 456, 874, 961, 1170, 1216, 1217, 1490, 151, 327, 456,
457, 493, 614, 695, 797, 1136, 1217, 1282, 1334, 1511], 'r': [35, 61, 94,
239, 374, 431, 503, 509, 724, 740, 758, 779, 807, 914, 981, 1051, 1077,
1103, 1261, 1263, 1366, 1509, 1514, 1605, 1685, 1688, 1690, 171, 258, 313,
437, 463, 497, 530, 704, 739, 790, 829, 856, 957,974, 997, 1146, 1236,
1264, 1408, 1417, 1439, 1519, 1615, 1619, 78, 141, 159, 302, 362, 399, 435,
 436, 437, 497, 564, 640, 658, 733, 739, 782, 990, 1207, 1210, 1286, 1397,
1418, 1429, 1519, 1530, 1740], 's': [36, 120, 131, 230, 267, 295, 309, 440,
 471, 499, 536, 660, 767, 819, 845, 933, 1057, 1093, 1127, 1303, 1311,
1314, 1365, 1403, 1501, 1660, 1699, 1804, 1851, 1858, 1890, 48, 84, 513,
534, 535, 661, 688, 788, 1303, 1528, 1750, 1881, 1887, 58, 118, 183, 268,
348, 373, 388, 653, 730, 799, 846, 962, 1000, 1135, 1184, 1360, 1474, 1528,
 1587, 1747, 1844, 1884], 'sh': [37, 142, 460, 820, 1137, 1238, 1239, 1399,
 80, 264, 728, 1415, 1416, 1438, 80, 337, 516, 642, 1001, 1022, 1037,
1694], 't': [38, 385, 529, 947, 1080, 1197, 1228, 1410, 1590, 1788, 121,
272, 507, 590, 684, 701, 1056, 1099, 1404, 1533, 1788, 90, 204, 271, 312,
427, 446, 465, 506, 709, 848, 850, 1017, 1215, 1788], 'th': [39, 787, 1123,
 1873, 361, 873, 927, 214, 361, 777, 1031, 1255, 1413], 'uh': [711, 1234,
1235, 1357, 40, 1111, 1234, 1744, 40, 242, 533, 632, 1234, 1744], 'uw':
[41, 199, 238, 405, 668, 671, 729, 775, 986, 1020, 1047, 1098, 1222, 1614,
1822, 93, 140, 371, 671, 732, 1221, 1327, 1328, 1384, 1392, 1419, 1491,
1640, 1768, 195, 307, 401, 451, 475, 532, 596, 774, 786, 855, 866, 1030,
1039, 1219, 1295, 1322, 1328, 1361, 1384, 1392, 1589, 1613, 1752, 1753,
1827, 1846, 1862, 1863], 'v': [42, 226, 703, 780, 843, 982, 1266, 1297,
1422, 1738, 1786, 229, 555, 556, 825, 998, 1069, 1246, 1554, 1562, 1759,
1799, 163, 289, 395, 438, 575, 692, 897, 1151, 1160, 1246, 1466, 1522,
1574, 1724], 'vcl': [43, 97, 282, 568, 650, 710, 747, 930, 993, 1064, 1130,
 1139, 1355, 1442, 1480, 1591, 1598, 1620, 1624, 1668, 1796, 1818, 150,
201, 301, 349, 409, 459, 548, 569, 629, 706, 890, 1033, 1112, 1128, 1272,
1353, 1481, 1576, 1621, 1623, 1624, 1798, 96, 200, 223, 285, 297, 324, 384,
 641, 679, 712, 723, 795, 798, 928, 929, 952, 1134, 1193, 1285, 1588, 1622,
 1764], 'w': [44, 133, 476, 543, 637, 714, 725, 1027, 1163, 1265, 1302,
1565, 1611, 1813, 85, 221, 305, 450, 527, 539, 675, 813, 870, 948, 1405,
1526, 1650, 113, 210, 353, 524, 620, 808, 939, 1073, 1166, 1245, 1280,
1435, 1651], 'y': [45, 208, 400, 473, 1592, 1698, 1720, 1780, 1868, 154,
224, 489, 677, 809, 1440, 1778, 1867, 488, 707, 1488, 1741, 1742], 'z':
[46, 269, 304, 310, 644, 987, 1050, 1159, 1191, 1259, 1427, 1433, 1482,
1777, 1807, 1816, 73, 128, 176, 439, 500, 776, 881, 904, 1118, 1445, 1672,
1749, 1776, 1811, 65, 156, 311, 390, 419, 428, 923, 953, 988, 1292, 1337,
1379, 1489, 1671, 1705, 1793, 1821], 'zh': [47, 496, 1853, 972, 1853]}

phonemes = ['sil', 'aa', 'ae', 'ah', 'ao', 'aw', 'ax', 'ay', 'b', 'ch', 'cl', '
```

```
    d', 'dh', 'dx', 'eh', 'el', 'en', 'epi', 'er', 'ey', 'f', 'g', 'hh', 'ih',
    'ix', 'iy', 'jh', 'k', 'l', 'm', 'n', 'ng', 'ow', 'oy', 'p', 'r', 's', 'sh'
    , 't', 'th', 'uh', 'uw', 'v', 'vcl', 'w', 'y', 'z', 'zh']



#Triphones
tritargets = {}
nTriTargets = 0
for n in range(1, 31):
    with open('tri3_ali.'+str(n)+'.pdf.txt') as f:
        triali = [x.strip() for x in f.readlines()]
    for item in triali:
        data = item.split()
        numdata = np.array([int(el) for el in data[1:]])
        tritargets[data[0]] = numdata
        nTriTargets = np.max([nTriTargets, numdata.max()])

nTriTargets += 1


trival = {}
nTriVal = 0
for n in range(1, 2):
    with open('tri3_ali_dev.'+str(n)+'.pdf.txt') as f:
        triali = [x.strip() for x in f.readlines()]
    for item in triali:
        data = item.split()
        numdata = np.array([int(el) for el in data[1:]])
        trival[data[0]] = numdata
        nTriVal = np.max([nTriVal, numdata.max()])

nTriVal += 1

tritest = {}
nTriTest = 0
for n in range(1, 2):
    with open('tri3_ali_test.'+str(n)+'.pdf.txt') as f:
        triali = [x.strip() for x in f.readlines()]
    for item in triali:
        data = item.split()
        numdata = np.array([int(el) for el in data[1:]])
        tritest[data[0]] = numdata
        nTriTest = np.max([nTriTest, numdata.max()])
nTriTest += 1


def frameConcat(x,splice, splType):
    validFrm = int( np.sum(np.sign( np.sum( np.abs(x), axis=1) )) )
    nFrame, nDim = x.shape

    if ( splType == 1):
        spl = splice
        splVec = np.arange(0, int(2*spl+1), 1)
    else:
        spl = int(2*splice)
        splVec = np.arange(0, int(2*spl+1), 2)

    xZerosPad = np.vstack([np.zeros((spl, nDim)), x[0:validFrm ,:], np.zeros((
        spl, nDim))])
    xConc = np.zeros( (validFrm, int(nDim*(2*splice+1))) )

    for iFrm in range(validFrm):
       xConcTmp = np.reshape(xZerosPad[iFrm+splVec,:], (1,int((2*splice+1)*
           nDim)) )
       xConc[iFrm, :] = xConcTmp
    return xConc

#13 MFCC:
x_tri = np.zeros((0, 143)) #13*11, 5 frames on each side of the current mfcc
x_test = np.zeros((0, 143))
x_val = np.zeros((0, 143))
y_test = np.zeros((0, nTriTest))
y_val = np.zeros((0, nTriVal))


trimodel = Sequential()
trimodel.add(Dense(512, activation='relu', input_shape=(143,)))
trimodel.add(Dense(512, activation='relu'))
trimodel.add(Dropout(0.25))
BatchNormalization(axis=1)
trimodel.add(Dense(512, activation='relu'))
trimodel.add(Dropout(0.25))
trimodel.add(Dense(nTriTargets, activation='softmax'))
trimodel.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['
    accuracy'])
trimodel.summary()
```

```
for keys in mfcc_train.keys():
    mfccarray = mfcc_train[keys]
    x_mean = np.mean(mfccarray, axis=0)
    x_std = np.std(mfccarray, axis=0)
    mfcctrain_normalized = ( mfccarray - x_mean ) / x_std
    trainConc=frameConcat(mfcctrain_normalized, 5, 1) #should give 13*11
    x_tri = np.vstack((x_tri, trainConc)) #concatenate mfcc

triarray = np.concatenate(list(tritargets.values()))


def tri_generator():
    used_so_far = 0
    batch_size=12929
    OHE = K.one_hot(triarray, nTriTargets)
    while True:
        if (used_so_far < (len(x_tri)-batch_size)+1):
            x_batch = x_tri[used_so_far:(used_so_far + batch_size), :]
            y_batch = OHE[used_so_far:(used_so_far + batch_size), :]
            yield(x_batch, y_batch)
            used_so_far += batch_size
        else:
            used_so_far = 0



for keys in mfcc_val.keys():
        valarray = mfcc_val[keys]
        mfccval_normalized = (valarray - x_mean ) / x_std
        valConc=frameConcat(mfccval_normalized, 5, 1)
        x_val = np.vstack((x_val, valConc))

        target_trival = trival[keys]
        Labels_val = np.eye(nTriVal)
        val_OHE = Labels_val[target_trival, :]
        y_val = np.vstack((y_val, val_OHE))

for keys in mfcc_test.keys():
    testarray = mfcc_test[keys]
    mfcctest_normalized = ( testarray - x_mean ) / x_std
    testConc=frameConcat(mfcctest_normalized, 5, 1)
    x_test = np.vstack((x_test, testConc))

    tri_test = tritest[keys]
    Labels_test  = np.eye(nTriTest)
    test_OHE = Labels_test[tri_test ,  :]
    y_test  = np.vstack((y_test,  test_OHE))



callback  = tf.keras. callbacks .EarlyStopping(monitor='val_loss', patience=6)
history  = trimodel. fit_generator (generator=(tri_generator()) ,  steps_per_epoch=
    np.ceil(len(x_tri )/12929), epochs=20, callbacks=[callback],
    use_multiprocessing=False,  validation_data=(x_val,  y_val) ,   shuffle =True)

#Test model: (on full  data set)
score,  acc  = trimodel.evaluate(x_test ,  y_test ,  verbose=0)
print('Test  score: ',  score)
print('Test  accuracy:',  acc)


targetClass  =  np.concatenate(list( tritest .values()))
predictedClass  = trimodel. predict_classes ( x_test )


#MAKES LIST OF LIST FROM DICTIONARY
statesValues  =  []
for  keys  in  states .keys():
    phonemeStates = states[keys] #this is a  list
    statesValues .append(phonemeStates)


### PHONEME RECOGNITION TEST SET ###

stateInPhoneme = []
correctIndexStates  = []
for  i  in range(len(x_test)):
    if  (predictedClass[ i ]  != targetClass [ i ]) :
        for index,   nested_list   in enumerate(statesValues):
            if predictedClass[ i ]   in   nested_list   and targetClass[i]  in
                 nested_list  :
                stateInPhoneme.append(i)

#Recalculating accuracy
correct  = 0
for j  in range(len(x_test)):
    if predictedClass[ j ]   == targetClass[j]:
        correct  +=1
```

```
correctPhonemes = len(stateInPhoneme)+correct


#Phoneme recognition:
newAccuracy = 100 * (correctPhonemes/len(predictedClass))
print('Phoneme recognition accuracy TEST SET: ', newAccuracy)

### END PHONEME RECOGNITION ###

################### SSL: STUDENT ###
predTarget =trimodel.predict(x_tri)

trimodel.save('modelTrimono.h5')
del trimodel

student2 = load_model('modelTrimono.h5')
student2.summary()

def student_generator():
    used_so_far = 0
    batch_size=12929
    while True:
        if (used_so_far < (len(x_val)-batch_size)+1):
            x_batch = x_tri[used_so_far:(used_so_far + batch_size), :]
            y_batch = predTarget[used_so_far:(used_so_far + batch_size)]
            yield(x_batch, y_batch)
            used_so_far += batch_size
        else:
            used_so_far = 0


callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=6)
history2 = student2.fit_generator(generator=(student_generator()),
    steps_per_epoch=np.ceil(len(x_tri)/12929), epochs=20, use_multiprocessing
    =False, validation_data=(x_val, y_val), callbacks=[callback] , shuffle=
    True)


score, acc = student2.evaluate(x_test, y_test)
print('Test score: ', score)
print('Test accuracy: ', acc)
```

```
### PHONEME RECOGNITION SSL ###
predictedClass = student2.predict_classes(x_test)
stateInPhoneme = []
for i in range(len(y_test)):
    if (predictedClass[i] != targetClass[i]):
        for index, nested_list in enumerate(statesValues):
            if predictedClass[i] in nested_list and targetClass[i] in
                 nested_list :
                stateInPhoneme.append(i)
                #print(index)

#Recalculating accuracy
correct = 0
for j in range(len(y_test)):
    #for j in range(len(x_train)):
    if predictedClass[j] == targetClass[j]:
        correct +=1
print('correct: ', correct)

correctPhonemes = len(stateInPhoneme)+correct
print(correctPhonemes)

#Phoneme recognition:
newAccuracy = 100 * (correctPhonemes/len(predictedClass))
print('Phoneme recognition accuracy: ', newAccuracy)

posPhoneme = []
posTarget = []
for i in range(len(predictedClass)):
    for index, nested_list in enumerate(statesValues):
        if predictedClass[i] in nested_list :
            posPhoneme.append(index)

        if targetClass[i] in nested_list :
            posTarget.append(index)

y_pred = posPhoneme
y = posTarget

cm=confusion_matrix(y, y_pred)
```

C fMLLR and monophones

fMLLR features and monophone targets

```
from kaldiio import ReadHelper

#Show full numpy array
np.set_printoptions(threshold=sys.maxsize)


train = dict()
with ReadHelper('scp:data-fmllr-tri3/train/feats.scp') as reader:
    for key, feats in reader:
        train[key] = feats


test = dict()
with ReadHelper('scp:data-fmllr-tri3/test/feats.scp') as reader:
    for key, feats in reader:
        test[key] = feats


dev = dict()
with ReadHelper('scp:data-fmllr-tri3/dev/feats.scp') as reader:
    for key, feats in reader:
        dev[key] = feats

#Monophones
#Targets for the training data:
targets = {}
nTargets = 0
for n in range(1, 31):
    with open('mono_ali.'+str(n)+'.pdf.txt') as f:
        monoali1 = [x.strip() for x in f.readlines()]
    for item in monoali1:
        #print(item)
        data = item.split()
        numdata = np.array([int(el) for el in data [1:]])
        targets [data [0]]  = numdata
        nTargets = np.max([nTargets, numdata.max()])

nTargets += 1

devTargets = {}
dev_nTargets = 0

for n in range(1,2):
    #Targets for the validation  data:
    with open('mono_ali_dev.'+str(n)+'.pdf.txt') as f:
        monoali_dev = [x.strip()  for x in f.readlines()]
    for item in monoali_dev:
        data_dev = item.split()
        numdata_dev = np.array([int(el) for el in data_dev [1:]])
        devTargets[data_dev[0]]  = numdata_dev
        dev_nTargets = np.max([dev_nTargets, numdata_dev.max()])

dev_nTargets  += 1

#Targets for the test  data:
testTargets  = {}
test_nTargets  = 0
for n in range(1,2):
    with open('mono_ali_test.'+str(n)+'.pdf.txt')  as  f:
        monoali_test  = [x. strip ()  for x in f. readlines () ]
    for item in monoali_test:
        data_test  = item. split ()
        numdata_test = np.array([int(el) for  el  in data_test  [1:]])
        testTargets [data_test [0]]  = numdata_test
        test_nTargets  = np.max([test_nTargets, numdata_test.max()])

test_nTargets  += 1


phonemes = ['sil', 'aa', 'ae', 'ah', 'ao', 'aw', 'ax', 'ay', 'b', 'ch', 'cl', '
    d', 'dh', 'dx', 'eh', 'el', 'en', 'epi', 'er', 'ey', 'f', 'g', 'hh', 'ih',
    'ix', 'iy', 'jh', 'k', 'l', 'm', 'n', 'ng', 'ow', 'oy', 'p', 'r', 's', 'sh'
    , 't', 'th', 'uh', 'uw', 'v', 'vcl', 'w', 'y', 'z', 'zh']




def frameConcat(x,splice, splType):
    validFrm = int( np.sum(np.sign( np.sum( np.abs(x), axis=1) )) )
    nFrame, nDim = x.shape

    if ( splType == 1):
        spl = splice
        splVec = np.arange(0, int(2*spl+1), 1)
    else:
        spl = int(2*splice)
        splVec = np.arange(0, int(2*spl+1), 2)

    xZerosPad = np.vstack([np.zeros((spl, nDim)), x[0:validFrm ,:],  np.zeros((
        spl, nDim))])
    xConc = np.zeros( (validFrm, int(nDim*(2*splice+1))) )

    for iFrm in range(validFrm):
        xConcTmp = np.reshape(xZerosPad[iFrm+splVec,:], (1,int((2*splice+1)*
            nDim)) )
        xConc[iFrm, :] = xConcTmp
    return xConc


model = Sequential()
model.add(Dense(440, activation='relu', input_shape=(440,)))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(256, activation='relu'))
model.add(Dense(nTargets, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['
    accuracy'])


model.summary()

#40-dimensional fMLLR:
x_train = np.zeros((0, 440)) #40*11, 5 frames on each side of the current mfcc
x_test = np.zeros((0, 440))
x_dev = np.zeros((0, 440))
y_train = np.zeros((0, nTargets))
y_test = np.zeros((0, test_nTargets))
y_dev = np.zeros((0, dev_nTargets))


for keys in train.keys():
    fmllr = train[keys]
    x_mean = np.mean(fmllr, axis=0)
    x_std = np.std(fmllr, axis=0)
    train_normalized = ( fmllr − x_mean ) / x_std
    trainConc=frameConcat(train_normalized, 5, 1) #should give 13*11
    x_train = np.vstack((x_train, trainConc)) #concatenate mfcc

    targetsarray = targets[keys]
    #numberOfClasses = np.max(targetsarray)+1
    Labels = np.eye(nTargets)
    targetOneHot = Labels[targetsarray, :]
    y_train = np.vstack((y_train, targetOneHot)) #concatenated targets


for keys in dev.keys():
    devarray = dev[keys]
    dev_normalized = ( devarray − x_mean ) / x_std
    devConc=frameConcat(dev_normalized, 5, 1)
    x_dev = np.vstack((x_dev, devConc))

    targetsarray_dev = devTargets[keys]
    Labels_dev = np.eye(dev_nTargets)
    targetOneHot_dev = Labels_dev[targetsarray_dev, :]
    y_dev = np.vstack((y_dev, targetOneHot_dev))


for keys in test.keys():
    testarray = test[keys]
    test_normalized = ( testarray − x_mean ) / x_std
    testConc=frameConcat(test_normalized, 5, 1)
    x_test = np.vstack((x_test, testConc))

    targetsarray_test = testTargets[keys]
    Labels_test = np.eye(test_nTargets)
    targetOneHot_test = Labels_test[targetsarray_test, :]
    y_test = np.vstack((y_test, targetOneHot_test))


callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=6)
history = model.fit(x_train, y_train, validation_data=(x_dev, y_dev),
      batch_size=256, epochs=20, callbacks=[callback], verbose=1, shuffle=True)
numberOfEpochs = len(history.history['loss'])


score, acc = model.evaluate(x_test, y_test, batch_size=256, verbose=0)
print('Test score:', score)
print('Test accuracy:', acc)
```

```
### PHONEME RECOGNITION ###
targetClass = np.where(y_test==1)[1]
predictedClass = model.predict_classes(x_test)


mapedState = [0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 7,
     7, 7, 8, 8, 8, 9, 9, 9, 10, 10, 10, 11, 11, 11, 12, 12, 12, 13, 13, 13,
    14, 14, 14, 15, 15, 15, 16, 16, 16, 17, 17, 17, 18, 18, 18, 19, 19, 19, 20,
     20, 20, 21, 21, 21, 22, 22, 22, 23, 23, 23, 24, 24, 24, 25, 25, 25, 26,
    26, 26, 27, 27, 27, 28, 28, 28, 29, 29, 29, 30, 30, 30, 31, 31, 31, 32, 32,
     32, 33, 33, 33, 34, 34, 34, 35, 35, 35, 36, 36, 36, 37, 37, 37, 38, 38,
    38, 39, 39, 39, 40, 40, 40, 41, 41, 41, 42, 42, 42, 43, 43, 43, 44, 44, 44,
     45, 45, 45, 46, 46, 46, 47, 47, 47]


#MAKES LIST OF LIST FROM DICTIONARY
statesValues = []
for keys in states.keys():
    phonemeStates = states[keys] #this is a  list
    #print(phonemeStates)
    statesValues.append(phonemeStates)



counter = 0
for i in range(len(y_test)):
    if (predictedClass[i] != targetClass[i]):
        for index, nested_list in enumerate(statesValues):
            if predictedClass[i] in nested_list and targetClass[i] in
                 nested_list :
                counter += 1

#Recalculating accuracy
correct = 0
for j in range(len(y_test)):
    if predictedClass[j] == targetClass[j]:
        correct +=1

correctPhonemes = counter+correct

newAccuracy = 100 * (correctPhonemes/len(x_test))
print('Phoneme recognition accuracy: ', newAccuracy)

### END PHONEME RECOGNITION
### SSL Student network ###

predicted = model.predict(x_train, batch_size=256, verbose=1)

model.save('modelfmllr.h5')
del model


student = load_model('modelfmllr.h5')
student.summary()

callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=6)
history1 = student.fit(x=x_train, y=predicted, validation_data=(x_dev, y_dev),
      batch_size=256, epochs=20, callbacks=[callback], verbose=1, shuffle=True)

score, acc = student.evaluate(x_test, y_test)
print('Test score: ', score)
print('Test accuracy: ', acc)



### PHONEME RECOGNITION SSL ####
predictedClass = student.predict_classes(x_test)

stateInPhoneme = []
for i in range(len(y_test)):
    if (predictedClass[i] != targetClass[i]):
        for index, nested_list in enumerate(statesValues):
            if predictedClass[i] in nested_list and targetClass[i] in
                 nested_list :
                stateInPhoneme.append(i)

correct = 0
for j in range(len(y_test)):
    if predictedClass[j] == targetClass[j]:
        correct +=1
print('correct: ', correct)


correctPhonemes = len(stateInPhoneme)+correct
print(correctPhonemes)

newAccuracy = 100 * (correctPhonemes/len(predictedClass))
print('Phoneme recognition accuracy: ', newAccuracy)
```

```
### END PHONEME RECOGNITION SSL ###

### CONFUSION MATRIX ###


posPhoneme = []
posTarget = []
for i in range(len(y_test)):
        for index, nested_list in enumerate(statesValues):
            if predictedClass[i] in nested_list:
                posPhoneme.append(index)
            if targetClass[i] in nested_list:
                posTarget.append(index)


y_pred = posPhoneme
y = posTarget


cm=confusion_matrix(y, y_pred)
```

D fMLLR and triphones

```
#Show full numpy array
np.set_printoptions(threshold=sys.maxsize)


train = dict()
with ReadHelper('scp:data-fmllr-tri3/train/feats.scp') as reader:
    for key, feats in reader:
        train[key] = feats

test = dict()
with ReadHelper('scp:data-fmllr-tri3/test/feats.scp') as reader:
    for key, feats in reader:
        test[key] = feats


val = dict()
with ReadHelper('scp:data-fmllr-tri3/dev/feats.scp') as reader:
    for key, feats in reader:
        val[key] = feats


tritargets = {}
nTriTargets = 0
for n in range(1, 31):
    with open('tri3_ali.'+str(n)+'.pdf.txt') as f:
         triali = [x.strip() for x in f.readlines()]
    for item in triali :
        numdata = np.array([int(el) for el in data[1:]])
        tritargets[data[0]] = numdata
        nTriTargets = np.max([nTriTargets, numdata.max()])

nTriTargets += 1


trival = {}
nTriVal = 0
for n in range(1, 2):
    with open('tri3_ali_dev.'+str(n)+'.pdf.txt') as f:
        triali = [x.strip() for x in f.readlines()]
    for item in triali:
        data = item.split()
        numdata = np.array([int(el) for el in data[1:]])
        trival[data[0]] = numdata
        nTriVal = np.max([nTriVal, numdata.max()])

nTriVal += 1

tritest = {}
nTriTest = 0
for n in range(1, 2):
    with open('tri3_ali_test.'+str(n)+'.pdf.txt') as f:
        triali = [x.strip() for x in f.readlines()]
    for item in triali:
        data = item.split()
        numdata = np.array([int(el) for el in data[1:]])
        tritest[data[0]] = numdata
        nTriTest = np.max([nTriTest, numdata.max()])

nTriTest += 1

ef frameConcat(x,splice, splType):
    validFrm = int( np.sum(np.sign( np.sum( np.abs(x), axis=1) )) )
    nFrame, nDim = x.shape

    if ( splType == 1):
        spl = splice
        splVec = np.arange(0, int(2*spl+1), 1)
    else:
        spl = int(2*splice)
        splVec = np.arange(0, int(2*spl+1), 2)

    xZerosPad = np.vstack([np.zeros((spl, nDim)), x[0:validFrm,:], np.zeros((
        spl, nDim))])
    xConc = np.zeros( (validFrm, int(nDim*(2*splice+1))) )

    for iFrm in range(validFrm):
       xConcTmp = np.reshape(xZerosPad[iFrm+splVec,:], (1,int((2*splice+1)*
           nDim)) )
       xConc[iFrm, :] = xConcTmp
    return xConc

#13 MFCC:
```

```
x_tri  = np.zeros((0, 440)) #13*11, 5 frames on each side of the current mfcc
x_test  = np.zeros((0, 440))
x_val  = np.zeros((0, 440))
y_test  = np.zeros((0, nTriTest))
y_val  = np.zeros((0, nTriVal))


trimodel = Sequential()
trimodel.add(Dense(512, activation='relu', input_shape=(440,)))
trimodel.add(Dense(512, activation='relu'))
trimodel.add(Dropout(0.25))
BatchNormalization(axis=1)
trimodel.add(Dense(512, activation='relu'))
trimodel.add(Dropout(0.25))
trimodel.add(Dense(nTriTargets, activation='softmax'))
trimodel.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['
    accuracy'])
trimodel.summary()



for keys in train.keys():
    fmllrarray  = train[keys]
    x_mean = np.mean(fmllrarray, axis=0)
    x_std  = np.std(fmllrarray, axis=0)
    train_normalized = ( fmllrarray  - x_mean ) / x_std
    trainConc=frameConcat(train_normalized, 5, 1)
    x_tri  = np.vstack((x_tri, trainConc)) #concatenate mfcc


triarray  = np.concatenate(list( tritargets . values()))

def tri_generator ():

    used_so_far  = 0
    batch_size=12929
    OHE = K.one_hot(triarray, nTriTargets)
    while True:
        if (used_so_far < (len(x_tri)-batch_size)+1):
            x_batch = x_tri[used_so_far :( used_so_far  + batch_size),  :]
            y_batch = OHE[used_so_far:(used_so_far + batch_size),  :]
            yield(x_batch, y_batch)
            used_so_far  += batch_size
        else:
            used_so_far  = 0

for keys in val.keys():
        valarray  = val[keys]
        val_normalized = (valarray  − x_mean ) / x_std
        valConc=frameConcat(val_normalized, 5, 1)
        x_val  = np.vstack((x_val,  valConc))

        target_trival   = trival [keys]
        Labels_val  = np.eye(nTriVal)
        val_OHE = Labels_val[target_trival,   :]
        y_val  = np.vstack((y_val,  val_OHE))

history  = trimodel. fit_generator (generator=(tri_generator()),  steps_per_epoch=
    np.ceil(len(x_tri)/12929), epochs=20, use_multiprocessing=False,
    validation_data=(x_val, y_val),   shuffle =True)


for keys in test.keys():
    testarray  = test[keys]
    test_normalized  = ( testarray  − x_mean ) / x_std
    testConc=frameConcat(test_normalized, 5, 1)
    x_test  = np.vstack((x_test,  testConc))

    tri_test   = tritest [keys]
    Labels_test  = np.eye(nTriTest)
    test_OHE = Labels_test[tri_test ,  :]
    y_test  = np.vstack((y_test,  test_OHE))


#Test model: (on full  data set)
score,  acc = trimodel.evaluate(x_test ,  y_test ,  verbose=0)
print('Test score:',  score)
print('Test accuracy:',  acc)


targetClass  =  np.concatenate(list( tritest .values()))
predictedClass  = trimodel. predict_classes ( x_test )


#MAKES LIST OF LIST FROM DICTIONARY
statesValues  = []
for keys in states.keys():
    phonemeStates = states[keys] #this is a  list
    statesValues.append(phonemeStates)
```

```
### PHONEME RECOGNITION ####
stateInPhoneme = []
correctIndexStates = []
for i in range(len(x_test)):
    if (predictedClass[i] != targetClass[i]):
        for index, nested_list in enumerate(statesValues):
            if predictedClass[i] in nested_list and targetClass[i] in
                nested_list :
               stateInPhoneme.append(i)

#Recalculating accuracy
correct = 0
for j in range(len(x_test)):
    if predictedClass[j] == targetClass[j]:
        correct +=1

correctPhonemes = len(stateInPhoneme)+correct

newAccuracy = 100 * (correctPhonemes/len(predictedClass))
print('Phoneme recognition accuracy TEST SET: ', newAccuracy)


###END PHONEME RECOGNITION ###



### SSL: Student model ###

predTarget =trimodel.predict(x_tri)

trimodel.save('modelTrifmllr.h5')
del trimodel


studentS = load_model('modelTrifmllr.h5')
studentS.summary()

def student_generator():
    used_so_far = 0
    batch_size=12929
    while True:
        if (used_so_far < (len(x_val)-batch_size)+1):
            x_batch = x_tri[used_so_far:(used_so_far + batch_size), :]
            y_batch = predTarget[used_so_far:(used_so_far + batch_size)]
            yield(x_batch, y_batch)
            used_so_far += batch_size
        else:
            used_so_far = 0


history2 = studentS.fit_generator(generator=(student_generator()),
    steps_per_epoch=np.ceil(len(x_tri)/12929), epochs=20, use_multiprocessing
    =False, validation_data=(x_val, y_val), shuffle=True)

score, acc = studentS.evaluate(x_test, y_test)
print('Test score: ', score)
print('Test accuracy: ', acc)

### PHONEME RECOGNITION SSL ###
predictedClass = studentS.predict_classes(x_test)

stateInPhoneme = []
for i in range(len(y_test)):
    if (predictedClass[i] != targetClass[i]):
        for index, nested_list in enumerate(statesValues):
            if predictedClass[i] in nested_list and targetClass[i] in
                 nested_list :
                stateInPhoneme.append(i)
                #print(index)

#Recalculating accuracy
correct = 0
for j in range(len(y_test)):
    if predictedClass[j] == targetClass[j]:
        correct +=1
print('correct: ', correct)

correctPhonemes = len(stateInPhoneme)+correct

newAccuracy = 100 * (correctPhonemes/len(predictedClass))
print('Phoneme recognition accuracy: ', newAccuracy)

### CM ###
posPhoneme = []
posTarget = []
```

```
for i in range(len(predictedClass)):
    for index, nested_list in enumerate(statesValues):
        if predictedClass[i] in nested_list :
            posPhoneme.append(index)

        if targetClass[i] in nested_list :
            posTarget.append(index)


y_pred = posPhoneme[0:len(posTarget)]
y = posTarget

cm=confusion_matrix(y, y_pred)
```