Rochester Institute of Technology
RIT Digital Institutional Repository

Theses

5-2017

Dysarthric Speech Recognition and Offline Handwriting Recognition using Deep Neural Networks

Suhas Pillai
sbp3624@rit.edu

Follow this and additional works at: https://repository.rit.edu/theses

Recommended Citation
Pillai, Suhas, "Dysarthric Speech Recognition and Offline Handwriting Recognition using Deep Neural Networks" (2017). Thesis. Rochester Institute of Technology. Accessed from

This Thesis is brought to you for free and open access by the RIT Libraries. For more information, please contact repository@rit.edu.

Dysarthric Speech Recognition and Offline Handwriting Recognition using Deep Neural Networks

by

Suhas Pillai

THESIS

Presented to the Faculty of the B.Thomas Golisano College of Computing
and Information Sciences
Rochester Institute of Technology
in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science in Computer Science, Department of Computer Science

Rochester Institute of Technology

May 2017

Dysarthric Speech Recognition and Offline Handwriting Recognition using Deep Neural Networks

APPROVED BY

SUPERVISING COMMITTEE:

Dr. Raymond Ptucha, Supervisor

Dr. Zack Butler, Supervisor

Dr. Emily Prud'hommeaux, Reader

Acknowledgments

I would like to thank professor Ray Ptucha, who has been an amazing mentor, who has encouraged and motivated me all the way through my thesis and as a student at RIT. I would like to thank my family for their support and my father who was an inspiration for my thesis topic. Finally, I would like to thank my friend Divya Nayak for her patience and support through out this thesis.

Abstract

Dysarthric Speech Recognition and Offline Handwriting Recognition using Deep Neural Networks

Suhas Pillai, M.S.
Rochester Institute of Technology, 2017

Supervisor: Dr. Raymond Ptucha

Millions of people around the world are diagnosed with neurological disorders like Parkinsons, Cerebral Palsy or Amyotrophic Lateral Sclerosis. Due to the neurological damage as the disease progresses, the person suffering from the disease loses control of muscles, along with speech deterioration. Speech deterioration is due to neuro motor condition that limits manipulation of the articulators of the vocal tract, the condition collectively called as dysarthria. Even though dysarthric speech is grammatically and syntactically correct, it is difficult for humans to understand and for Automatic Speech Recognition (ASR) systems to decipher. With the emergence of deep learning, speech recognition systems have improved a lot compared to traditional speech recognition systems, which use sophisticated preprocessing techniques to extract speech features.

In this digital era there are still many documents that are handwritten many of which need to be digitized. Offline handwriting recognition involves recognizing handwritten characters from images of handwritten text (i.e. scanned documents). This is an interesting task as it involves sequence learning with computer vision. The task is more difficult than Optical Character Recognition (OCR), because handwritten letters can be written in virtually infinite different styles. This thesis proposes exploiting deep learning techniques like Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) for offline handwriting recognition. For speech recognition, we compare traditional methods for speech recognition with recent deep learning methods. Also, we apply speaker adaptation methods both at feature level and at parameter level to improve recognition of dysarthric speech.

Table of Contents

Acknowledgments **iii**

Abstract **iv**

List of Tables **viii**

List of Figures **ix**

Chapter 1. Introduction **1**

1.1 Research Summary 3

1.2 Novel Contributions 5

Chapter 2. Background **7**

2.1 Speaker adaptation using GMM-HMM 11

2.1.1 Speaker adaptation using Maximum Likelihood Estimation and Maximum a Posteriori 11

2.2 Speaker adaptation using Deep Neural Networks 15

2.2.1 Speaker adaptation using speaker codes 15

2.2.2 Rapid and Effective speaker adaptation of Convolutional Neural Network 16

2.2.3 Speaker adaptation using low rank matrix factorization 19

2.2.4 KL-Divergence regularized Deep Neural Networks 21

2.3 Handwriting Recognition 23

Chapter 3. Methodology **27**

3.1 DNN Framewise Training 27

3.1.1 Mel-Frequency Cepstral Coefficients 28

3.1.2 Linear Discriminant Analysis 29

3.1.3 Maximum Likelihood Linear Transformation 30

3.1.4 Feature space Maximum Likelihood Linear Regression . 31
3.2 CNN-RNN-CTC Training 35
3.2.1 Convolutional Neural Network 35
3.2.2 Recurrent Neural Network 37
3.2.3 Batch Normalization 38
3.2.4 Dropout 42
3.2.5 Connectionist Temporal Classification 43
3.2.6 Decoding 47
3.2.7 Learning Hidden Unit Contributions 48
3.3 Multidimensional Recurrent Neural Network 50
3.4 Dataset 55
3.4.1 Dysarthric Speech Recognition 55
3.4.2 Handwriting Recognition 58

Chapter 4. Results and Discussion **64**

Chapter 5. Conclusion and Future Work **76**

Bibliography **80**

List of Tables

3.1 Statistics of Dysarthric Speakers data 57
3.2 Statistics of Control Speakers data 57

4.1 CNN-RNN-CTC results in % WER. 67
4.2 GMM-HMM and DNN-HMM results in % WER. 70
4.3 % CER on Test set. 74

List of Figures

2.1 Spectrogram representation of *[i][a][u]* 8
2.2 Learning Speaker Codes [4]. 17
2.3 Preprocessing pipeline. 24

3.1 Feature extraction pipeline for DNN. 28
3.2 Difference Between Forced Alignment and CTC. 44
3.3 Forward Backward Algorithm for CTC. 46
3.4 Beam Search Decoding with CTC [41]. 49
3.5 MDLSTM unit architecture. 54

4.2 Filters of MDLSTM network. 75

Chapter 1

Introduction

Offline handwriting recognition is a more challenging problem than online handwriting recognition [51]. In online handwriting recognition, features can be inferred both from past and present, whereas in offline handwriting recognition, features can only be obtained using a still image. In both the cases, input features have traditionally been extracted from data, then a classifier like Artificial Neural Network (ANN) or Gaussian Mixture Model (GMM), are used to estimate posterior probabilities. These posterior probabilities are input to a Hidden Markov Model (HMM) to generate transcriptions. One of the major disadvantages of HMMs is that they fail to model long term dependencies in input data. By long term dependencies in offline context it is meant, given an input word or a sentence to recognize, how likely is a prediction of a character or a word depend on previous characters or words seen by the model. Recurrent Neural Networks (RNNs) with Long Short Term Memory (LSTM) units [25] can help to resolve this drawback. LSTMs can model long dependencies and have shown remarkable improvement in sequence learning tasks like speech recognition [21], machine translation [64], video summarization [62], and more.

One of the advantages of using deep neural networks is that inputs can be unprocessed data such as raw pixels of an image, rather than extracting specific features in previous methods [66]. Input to RNNs is usually 1D. For example, in online handwriting recognition, a common feature is pen stroke grid values. But in offline recognition, the input is a 2D image. A naive way would be taking every column of an image as a 1D vector and feeding it as an input to a RNN. However, this cannot handle distortions along vertical axis. The same image will look different if a pixel is shifted down by one pixel. Another way to tackle this problem is to use multidimensional RNNs, which take contextual information from all the directions, i.e left, right, top and bottom. The idea is to use both spatial and temporal information. The use of Connectionist Temporal Classification (CTC) enables inputs to be used without any prior segmentation as opposed to forcefully aligning inputs in previous approaches [20].

Ideas from handwriting recognition can be applied to dysarthric speech recognition. One of the major advantages of the CTC algorithm is that you do not need properly segmented labeled data. The CTC algorithm takes care of the alignment of input with the output. This is one of the major advantages because dysarthric speech is slurred, choppy or mumbled and to align or segment a speech waveform belonging to a particular phoneme (perceptually distinct unit of sound) is a challenging task and takes hours of manual labor. For traditional GMM-HMM systems on speech recognition, common input fea-

tures include Mel Frequency Cepstral Coefficients (MFCCs) [22], Perceptual Linear Predictive coefficients (PLPs) extracted from raw waveforms [14], and first and second order temporal differences. In this research, we try both features extracted from traditional feature processing pipelines and spectrograms as an input to two different architectures of deep neural networks. In this thesis, spectrogram refers to doing a Fast Fourier Transform (FFT) on a speech file, using sampling rate of 16 KHz, sampling window of 20ms or 15ms and a stride of 10ms to get energy representation for every sampling window. Stacking these energy representation for every window from left to right forms the spectrogram, as opposed to this usage in other disciplines where a spectrogram is a visual representation of speech.

1.1 Research Summary

- Input spectrograms to Convolutional Neural Network (CNN), Recurrent Neural Network (RNN) and Connectionist Temporal Classification (CTC) (i.e CNN-RNN-CTC) architecture.

- Sampled speech waveform with 20 ms and 15 ms window size for dysarthric speakers. Sampling speech with 20 ms window size and a stride of 10ms gave better performance.

- Trained CNN-RNN-CTC model on 1000 hours of normal speech and adapted model for dysarthric speech. For speaker adaptation, we tried fine tuning different layers of the network with different learning rates.

Learning rate of 0.0001 for fully connected layers and 0.0003 for rest of the network worked well. Extracting speaker codes for each speaker for rapid speaker adaptation was also tried but did not perform well.

- Adding Learning Hidden Unit contributions (LHU) layer after the recurrent layers worked well for speaker adaptation. Adding more than 1 LHU layer deteriorated the performance of the network. Also, adding LHU layer at the bottom gave better performance than adding LHU layer at the top.

- Adding dropout after all the layers did not improve the performance. Adding dropout after the 1st RNN layer improved performance. Adding dropout at convolutional layers with probability of 0.4 worked well than with probability of 0.5.

- For batch normalization deciding batch size is a hyper parameter, we tried batch size of 5, 10 and 30. Batch size of 30 did well on one of the dysarthric speakers but for others the accuracy was same as obtained using batch size of 10.

- Augmenting data for CNN-RNN-CTC model using speed perturbation, tempo perturbation and amplifying original speech boosted performance of the network.

- Implemented beam search decoding for decoding output predictions using Character Language Model (CLM). When tested on normal speech,

using beam search decoding reduces Word Error Rate (WER) by 6%. For dysarthric speakers, training and testing set is a combination of single words and long sentences. We found beam search decoding with CLM did not reduce WER % because with single words it does not get enough contextual information to correct the characters in the word. For long sentences, the dysarthric speaker WER % is on a higher side, so using beam search decoding with CLM does not improve accuracy or reduce WER %. The WER % should be less than 30 % for beam search decoding with CLM to be effective.

- We found for training models with features extracted using traditional methods, normalizing input speech features using Feature spaced Maximum Likelihood Linear Regression (fMLLR) works well on some dysarthric speakers, especially for female dysarthric speakers.

1.2 Novel Contributions

- Learning Hidden Unit (LHU) contributions layer after recurrent layer worked well for speaker adaptation.

- Data augmentation using speed perturbation, tempo perturbation and amplifying original speech boosted performance.

- When inter speaker variability is high and amount of data is less for training, features extracted using traditional methods followed by fMLLR adaptation works better than features extracted from CNNs.

- Trained CNN-RNN-CTC model on clean speech data from audio books and adapted model on dysarthric speech.

Chapter 2

Background

Some previous work on dysarthric speech intelligibility involved resynthesizing formants of dysarthric speech [30]. In [30], formants were analyzed to understand the acoustics reasons for impairment in dysarthric speech. [31] specifically analyzed the first two formant frequencies of vowels, along with duration and energy. A formant is a concentration of acoustic energy around a particular frequency in the speech wave. Based on source-filter theory of speech production, vocal tract filters a source sound (e.g. periodic voice vibrations or aperiodic hissing) and it is due to this filtering we can produce different sounds. Formants are seen on spectrograms, around frequencies that correspond to the resonances of the vocal tract (i.e at frequencies where resistance to vibration is low). There are several formants (i.e F1.....F9) and each occur at different frequency, roughly one in each 1000Hz band. Figure 2.1 shows three spectrograms for speech sounds [i], [a] and [u], where dark regions represent amount of energy at particular frequencies. For [i] the mean F1 is 761.66 Hz and mean F2 is 1663.86 Hz; for [a] the mean F1 is 467.29 Hz and mean F2 is 2135.94 Hz; and for [u] the mean F1 is 1125.30 Hz and mean F2 is 2233.18 Hz. We can see [i] has higher F1 and low F2 than [a], and [i] has low F1 and F2 than [u]. Thus, based on formant frequencies F1 and F2 all the

vowels can be characterized. Experiments were done on different combinations of F1, F2, F1 median, F2 median, F1 slope left, F1 slope right, where slope left and slope right refers to median first difference in formant frequency value during the first and last 30% of the vowel region. These features extracted from dysarthric speech waveforms were used to train Gaussian Mixture Models (GMM) to generate posterior probabilities. GMMs generate a mapping function, which transforms vowels of a speaker with dysarthria to closely match the vowel space of a non-dysarthric speaker. This improves intelligibility of vowels from 48% to 58%.

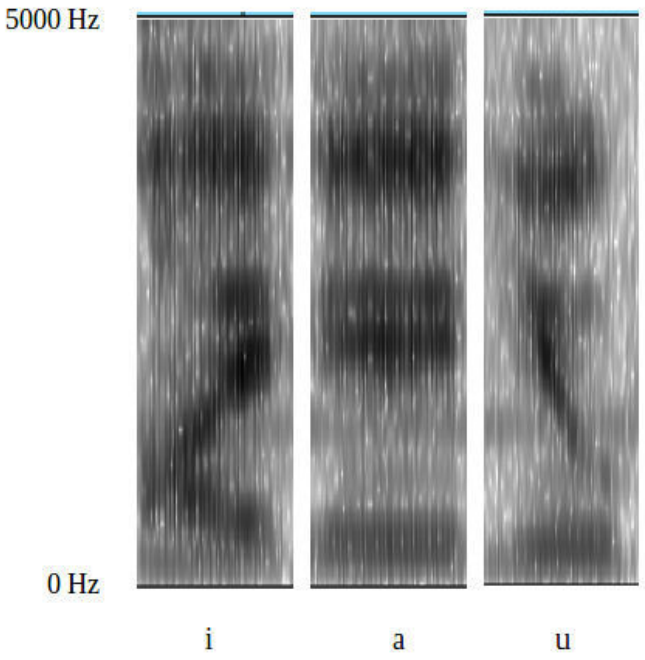

Figure 2.1: Spectrogram representation of *[i][a][u]*

Rudzicz [55] proposed modifications to dysarthric speech, which in-

volved correction of tempo, adjustment of formant frequencies in vowels, removal of abnormal or irregular voicing, deletion of phoneme and replacement of erroneously dropped phonemes. The duration of dysarthric speech is usually longer compared to clear speech, and there are phonemes that are inserted or deleted due to speech impairment. When an insertion is identified, the associated speech segment is removed. In the case were an associated segment is not surrounded by silence, adjacent phonemes are merged using Pitch Synchronous Overlap and Add (PSOLA). For the deletion of a phoneme, the speech segment from synthesized speech corresponding to the dysarthric phoneme is extracted and inserted into the dysarthric speech. When it is an unvoiced fricative, affricate or plosive, no further action is taken. For voiced phonemes, F0 curve from synthesized speech segment is extracted and removed. F0 represents fundamental frequency or pitch, it is the property of the source (i.e number of vocal fold vibrations) and perceived by the ear as pitch. Generally, F0 for men would be around 120 Hz and for women around 200 Hz [48]. The F0 curve is then linearly interpolated from adjacent phonemes of source dysarthric speech. If interpolation is not possible then a flat F0 equal to the nearest natural F0 curve is generated. Since, vowels uttered by dysarthric speaker are significantly slower than those uttered by typical speaker [56], using morphing in time, phonemes of dysarthric source are shortened to synthetic phoneme length. Morphing in frequency involves modifying formant trajectories of a vowel in dysarthric speech to known vowel identity of the speech segment.

Nakashika et al. [46] proposed a robust feature extraction method using

Convolutive Bottleneck layer Networks (CBN), where traditional MFCC features were replaced by convolutive bottle neck features. The input to CBNs is a mel-map, where Y-axis represents mel-frequency and X-axis is time. CBNs follows convolutional neural network architecture, where one of the feed forward layers are replaced by a bottle neck layer of smaller size (i.e. less number of hidden neurons). The idea behind this is same as Principal Component Analysis (PCA), to reduce the dimensionality [24] and decorrelate features to get a compact representation of features representing the given input signal. [46] tested their system on speech produced by persons with articulation disorders resulting from athetoid cerebral palsy, and found features extracted by CBN were better able to capture unstable speaking style caused by athetoid symptoms. The features extracted were used to train another HMM for speech recognition. Using CBN features reduced Word Error Rate (WER) by 4% over using MFCCs as features for speech recognition.

Most of the methods described above involve modifying speech and then using the modified speech for recognition, however this won't be scalable to larger systems with many users. Also, one of the problems for people with dysarthric speech is that their disease prevents them from recording speech for long periods of time. As a result, we don't have much data from speakers with dysarthric speech. In order to overcome this shortage of data, we adopt techniques from speaker adaptation in speech recognition. In speech recognition, we train the model on training data and test it on test data but in real world scenario we cannot cover all the different ways in which people say the same

word. There might be people, who have fast speaking styles or speakers with different accents, which might not be there in the training data. Such systems will do poorly on a new speaker, so in order to overcome this problem, the model tries to adapt to this new speaker using adaptation techniques. Following sections describe speaker adaptation techniques used in GMM-HMM and DNN-HMM systems.

2.1 Speaker adaptation using GMM-HMM

2.1.1 Speaker adaptation using Maximum Likelihood Estimation and Maximum a Posteriori

Gaussian Mixture Model (GMM) is a parametric probability density function represented as a weighted sum of gaussian component densities. GMMs are commonly used as a parametric model of the probability distribution of continuous measurements or features in a biometric system, such as vocal-tract related spectral features in a speaker recognition system. The parameters of GMM are estimated using iterative Expectation Maximization (EM) algorithm or Maximum A Posteriori (MAP) estimate from a well-trained prior model. GMM can be represented by the following equation

$$p(x|\lambda) = \sum_{i=1}^{M} w_i g(x|\mu_i, \sum_i) \tag{2.1}$$

where x is a D dimensional vector (i.e measurement of features), w_i i=1,...,M, are mixture weights and $g(x|\mu_i, \sum_i)$ i=1,...,M, are the component gaussian densities. Each component density is a D-variate function of the form.

$$g(x|\mu_i, \sum_i) = \frac{1}{(2\pi)^{D/2} \det \sum_i^{1/2}} \exp\{\frac{-1}{2}(x-\mu_i)' \sum_i^{-1} (x-\mu_i)\} \qquad (2.2)$$

where μ_i is a mean vector and $\sum_i$ is a covariance matrix. The component densities from GMM might represent some hidden classes. For example in speech recognition, the spectral features might represent speaker's phonetic events like vowel, fricatives etc.

To find parameters of GMM which is represented by λ that better estimates the distribution of training feature vectors, the most popular and well established method is Maximum Likelihood Estimation (MLE). The idea of MLE is to start with initial estimate of parameters, then refine the parameters using an iterative Expectation Maximization (EM) algorithm. Generally, the initial estimate of parameters is done using Vector Quantization (VQ). The ML estimate finds the model parameters that maximizes the likelihood of GMM, given the training data. For a set of T training vectors X = $X_1, X_2 ... X_T$ the GMM likelihood is given by

$$P(X|\lambda) = \prod_{i=1}^{T} P(x_i, \lambda) \qquad (2.3)$$

On each EM iteration, the following re-estimation formula is used which guarantees a monotonic increase in model's likelihood value.

$$w_i = \frac{1}{T} \sum_{t=1}^{T} Pr(i|x_t, \lambda) \qquad (2.4)$$

$$\mu_i = \frac{\sum_{t=1}^{T} Pr(i|x_t, \lambda) x_t}{\sum_{t=1}^{T} Pr(i|x_t, \lambda)} \tag{2.5}$$

$$\sigma_i^2 = \frac{\sum_{t=1}^{T} Pr(i|x_t, \lambda) x_t^2}{\sum_{t=1}^{T} Pr(i|x_t, \lambda)} - \mu_i^2 \tag{2.6}$$

$$Pr(i|x_t, \lambda) = \frac{w_i g(x_t|\mu_i, \sum_i)}{\sum_{k=1}^{M} w_k g(x_t|\mu_k, \sum_k)} \tag{2.7}$$

where $Pr(i|x_t, \lambda)$ is the posterior probability for i^{th} gaussian component

In addition to estimating parameters using EM algorithm, parameters can also be estimated using Maximum a Posteriori (MAP) estimation. This technique is very popular in speaker adaptation systems, where a universal background model (UBM) [54] is trained on data from many speakers, then this model is adapted to new speaker, where there is not enough labeled data for training. The expectation step remains the same as in EM algorithm described above, where sufficient statistics (i.e count , first and second order moments) are calculated for each mixture using prior model. While for adaptation these new mixture statistics i.e $(w_i, \mu_i, \sum_i)$ are combined with old mixture statistics from λ_{prior} using data dependent coefficients. These coefficients are designed such that mixtures with high counts rely more on new statistics for final parameter estimation and mixtures with low counts of data rely more on the old statistics for parameter estimation.

$$n_i = \sum_{t=1}^{T} Pr(i|x_t, \lambda_{prior}) \tag{2.8}$$

$$E_i(x) = \frac{\sum_{t=1}^{T} Pr(i|x_t, \lambda_{prior}) x_t}{n_i} \tag{2.9}$$

$$E_i(x^2) = \frac{\sum_{t=1}^{T} Pr(i|x_t, \lambda_{prior}) x_t^2}{n_i} \tag{2.10}$$

These new statistics from the training data are used to update the prior statistics for mixture i to create adapted parameters for mixture i. The adapted mixture weights, first order moment (mean) and second order moment (variance) are given as

$$w_i = [\alpha_i^w \frac{n_i}{T} + (1 - \alpha_i^w) w_i] \gamma \tag{2.11}$$

$$\mu_i^- = [\alpha_i^m E_i(x) + (1 - \alpha_i^m) \mu_i] \tag{2.12}$$

$$\sigma_i^{2-} = [\alpha_i^v E_i(x)^2 + (1 - \alpha_i^v)(\sigma_i^2 + \mu_i^2) - \mu_i^{2-}] \tag{2.13}$$

The adaption coefficients $\alpha_i^\tau \in (w, \mu, \sigma)$ control the balance between old and new estimates for weights, means and variances. The scaling factor γ is selected so that all the weights sum to unity. For each mixture the adapted coefficient $\alpha_i^\tau \in (w, \mu, \sigma)$ is calculated as follows

$$\alpha_i^\tau = \frac{n_i}{n_i + r^\tau} \tag{2.14}$$

where r^τ is a fixed relevance factor for parameter τ. Thus, when $\alpha_i^\tau \to 0$, then less importance is given to new estimates, while if $\alpha_i^\tau \to 1$, then less importance is given to old estimates.

2.2 Speaker adaptation using Deep Neural Networks

2.2.1 Speaker adaptation using speaker codes

As seen in the previous section, speaker adaptation techniques like Maximum a Posteriori estimation involve modifying model parameters to fit new speaker data. It has performed good, when large amount of adaptation data is available. Other methods like Vocal Tract Length Normalization (VTLN) [38] are also used for speaker adaptation, where we learn a parametric warping function that is used to normalize speech features. The method is computationally efficient and robust because it involves only one free parameter to be estimated per speaker. But its drawback is that the warping function has to be designed manually.

Ossama et al. [3] proposed speaker adaptation using a Hybrid NN-HMM model. The baseline hybrid NN-HMM model computes posterior probabilities of all the HMM states given each input feature vector. In hybrid NN-HMM the output classes are HMM states. Standard backpropagation algorithm is used to optimize weights of a NN, where cross entropy is used as an objective function. The proposed adaptation technique involves learning a

generic adaptation NN along with speaker specific codes. In their method, the adaptation layer is inserted below the original NN-HMM model, where all the layers in adaptation network are fully connected layers. Along with input from the lower layers, the adaptation NN also receives speaker specific codes called speaker codes as input as shown in Figure 2.2 (a). In their model, speaker specific codes are learned for every speaker S_i, which represents a compact feature vector representing speaker dependent information. The speaker code along with generic adaptation NN helps to transform new speaker's data into speaker independent feature space. The advantage of speaker code is that it reduces the amount of adaptation data required per speaker.

The training phase consists of learning three sets of parameters, 1. Learning normal NN weights without inserting adaptation NN, which results in Speaker Independent (SI) NN. SI NN model is learned on a training set. 2. Adaptation NN weights and speaker codes are learned on development set. 3. Speaker codes are learned using small amount of data from testing set for each speaker. This is a supervised adaptation method, where a small set of labeled utterances are available for adaptation. State level alignments are obtained from a trained HMM model using standard forced alignment method.

2.2.2 Rapid and Effective speaker adaptation of Convolutional Neural Network

Ossama [3] proposed rapid adaptation for DNN, where speaker adaptation consisted of three phases, 1. Learning normal NN weights without

inserting adaptation NN 2. Learning adaptation NN and 3. Learning speaker codes. The speaker codes along with adaptation NN helps to transform input features to a more speaker normalized space, where speaker differences are removed and speech recognition accuracy is improved. However, the proposed technique did not perform well on CNNs. This was mainly because of the difficulty in training the fully connected layer beneath the convolution layer. This layer involves the complexity of pooling and convolution operation that assume input is grouped in number of frequency bands, which is not the case in fully connected layers of adaptation DNN. In order to avoid this, adaptation NN is configured above convolution and pooling layer as shown in Figure 2.2 (b). As a result, the adaptation NN will transform the features computed by CNN layer instead of original input speech features.

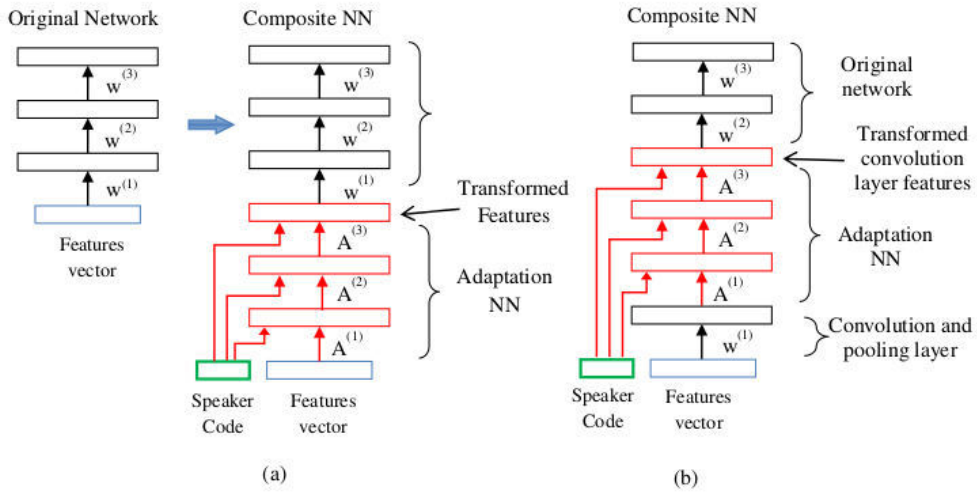

Figure 2.2: Learning Speaker Codes [4].

Along with speaker codes and adaptation NN, they also proposed a new adaptation method called adaptive nodes output weights. The idea is to adapt the NN to a target speaker by scaling the outputs after applying an activation function. In the case of the fully connected layer, this may be thought of as simply increasing or decreasing output nodes. In the case of the convolution layer, this may be thought of as forcing the pooling operation towards a more preferable frequency shift. [49] studied male and female acoustic differences in French and English speakers. Fundamental frequency mean F0 was much higher in female speakers in both the languages. Also, there was not much difference in formant frequency F1 between male and female English speakers but F2 and F3 formant frequencies showed significant difference between male and female English speakers. Thus, for a male speaker there may be tendency to decrease activation of nodes representing shifts towards high frequencies of the same feature map kernel [10]. Let's assume that the i_{th} node in a certain NN layer has the output o_i. The scaled output is then given as

$$o_i{}^{-} = o_i \exp(v_i) \tag{2.15}$$

where v_i is the weight that is multiplied with the output from layer 'l'. Here the weight is represented as an exponential of the parameter v_i in order to have positivity during training. The derivatives are computed using standard backpropagation algorithm.

Derivative with respect to o_i

$$\frac{\partial E}{\partial o_i} = \frac{\partial E}{\partial {o_i}^-} \quad \exp(v_i) \tag{2.16}$$

Derivative with respect to v_i

$$\frac{\partial E}{\partial v_i} = \frac{\partial E}{\partial {o_i}^-} o_i \quad \exp(v_i) \tag{2.17}$$

For a convolution layer the i_{th} feature map of j_{th} frequency band at s_{th} shift location after applying output weight is:

$$o^-_{i,j,s} = o_{i,j,s} \quad * \quad \exp(v_{i,j,s}) \tag{2.18}$$

Equation 2.18 shows that if at certain s_{th} shift, if $v_{i,j,s}$ is zero, then that particular region is not considered in the maximization operation. As a result, the maximization operation is forced towards certain frequency shifts depending on speaker characteristics.

2.2.3 Speaker adaptation using low rank matrix factorization

Normally, when we do speaker adaptation in DNN, we fine tune the parameters (i.e weight matrix) between last hidden layer and output layer, using small amount of new speaker's data. However, the output layer contains (typically between 2000 - 10000 output units) depending on context dependent states of GMM-HMM system. This results in many parameters (i.e weight matrix is huge) between last hidden layer and output layer. Gemello et al. [18] proposed Linear Hidden transformation, where an affine layer is added

between the last hidden layer and output layer. This layer is learned keeping all other parameters fixed. In order to reduce the number of parameters to learn between last hidden layer and output layer, [57] [72] [73] proposed bottleneck layer and learning only bottleneck layer parameters on the adaptation data. Adding a linear or affine bottleneck layer between the last hidden layer and output layer, also known as Linear Hidden Network (LHN) showed to have superior performance than Linear Input Network (LIN), where an affine layer is inserted between input and first hidden layer. If W_L is the weight between last hidden layer and output layer, then output Y_L is given by

$$Y_L = softmax(W_L Y_{L-1}) \tag{2.19}$$

Reducing number of parameters is done by replacing weight matrix W_L by a low rank matrix. If we denote the weight matrix of final layer as A = $m \times n$. If A has a rank r, then there exists a factorization [63] A = B $\times$ C, where B is full rank matrix of size $m \times r$ and C is full rank matrix of size $r \times n$. Thus, we want to replace matrix A by matrices B and C. There is no non linearity (i.e sigmoid, ReLU) between matrices B and C. We want the number of parameters in B(mr) and C (rn) to be less than A (mn) . Thus, if we want to reduce number of parameters by a fraction p, then we require following to hold true.

$$mr + rn < pmn \tag{2.20}$$

$$r < \frac{pmn}{m+n} \tag{2.21}$$

One of the ways to choose correct value of r is to check number of active output targets (i.e number of neurons having values greater than threshold 1e-3) for 40-50 utterances. The value of r can then be chosen as the number of active output units. Also, as the number of parameters reduces, this method also speeds up training.

2.2.4 KL-Divergence regularized Deep Neural Networks

Yu et al. [75] proposed an adaptation technique by forcing the estimated distribution from the adapted model to be close to an unadapted model. This is done by adding Kullback-Leibler divergence (KLD) regularization to the adaptation criterion. The idea is that adding the regularization term is the same as forcing the estimated distribution to be close to a target distribution. Adaptation to the new model can be done by using speaker independent model and learning weights on new speaker data. As weights of the model change, it might do poorly on the original data on which a speaker independent model was trained. To prevent this from happening, adaptation needs to be done conservatively. The intuition behind their approach is that the posterior of output (i.e senone or characters) distribution estimated from the adapted model should not deviate far away from the distribution of unadapted models, especially when adaptation set is small. As DNNs outputs are probability distributions, a natural choice in measuring the deviation is

the Kullback-Leibler divergence (KLD). Thus, by adding this divergence term to the cost function as a regularization term, we can prevent the model from overfitting. Suppose, the cost function is given by.

$$E = \frac{1}{N} \sum_{t=1}^{N} \sum_{y=1}^{S} p^{-}(y|x_t) \log(p(y|x_t)) \tag{2.22}$$

where N is number of training samples, x_t is t^{th} observation frame and $p^{-}(y|x_t)$ is target probability, which is 1 for correct class else 0. Now, adding KLD as a regularization term gives the following cost function:

$$E = \frac{1}{N} \quad p^{*}(y|x_t) log(p(y|x_t)) \tag{2.23}$$

$$p^{*}(y|x_t) = [(1-\rho)p^{-}(y|x_t) + \rho \log(p^{SI}(y|x_t))]$$

After adding KLD regularization, the original target distribution is changed from $p^{-}(y|x_t)$ to $p^{*}(y|x_t)$, which is a linear interpolation of the distribution estimated from an unadapted model and ground truth of the adaptation data. This interpolation prevents a new adapted model going far away from the SI model. This is different from L2 regularization, which constrains weights of the model,where as here output probabilities are constrained. Regularization weight ρ can be adjusted based on adaptation data. When $\rho = 1$, we do not adapt to new adaptation data, while if $\rho = 0$ we adapt the model to new adaptation data, ignoring previous SI model. Ideally, ρ should be large for small adaptation data, so that the model does not overfit on new adaptation data and small, if the adaptation set is large.

To summarize, the speaker adaptation techniques involve modifying model parameters (i.e in GMMs we modify mean, covariance and weights, while in neural networks we modify the weights of the network). In order to do this in an efficient way, we need to design algorithms that learn fewer adaptation parameters or that learn a rich representation of speakers given a small amount of new test speaker data.

2.3 Handwriting Recognition

Previous work on handwriting recognition involved complex preprocessing pipelines before classification or recognition. Figure 2.3 represents that workflow:

Preprocessing techniques enhance image rendering, making it more suitable for segmentation. This process involves binarization of an input image using a threshold. This can be followed by detection of edges on the binarized image. Morphological operations, such as dilation and erosion are also typically performed on images. Dilation adds pixels to the boundaries of the object in an image. The value of the output pixel is the maximum value of all the pixels in input neighborhood. For a binary image, if any pixel value is 1 in that set of neighbors, then the output pixel value is 1. Erosion removes pixels from object boundaries, the value of output pixel is minimum value of the neighboring pixel values. If any of the pixel values in the set are 0, then the output pixel value is 0. Noise removal is one of the topics that has been researched extensively for typed documents. The scanning process introduces noise and

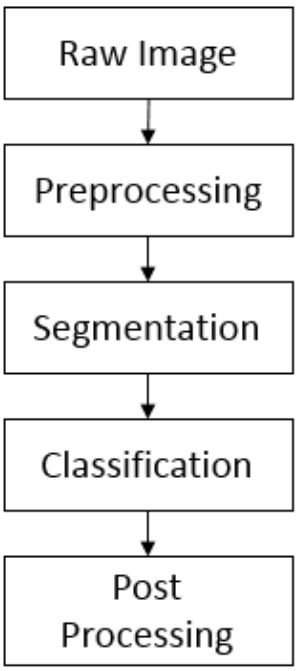

Figure 2.3: Preprocessing pipeline.

there are smoothing operations followed to remove artifacts introduced during image capture. Thinning operations are performed on offline images to infer strokes from neighboring text lines [45].

Recognizing handwritten characters using traditional approaches involves extracting features for classification, extracting features for segmentation, and parsing to map spatial relationships among characters for recognition. Segmenting of text into lines, words, and characters requires sophisticated approaches. Segmenting unconstrained handwritten words or characters is much more difficult than typed text because handwritten text can undulate up and

down. [26] described an approach to separate a line of unconstrained text to words. They proposed a gap metrics based approach to perform word segmentation. They extracted local features like distance between current pair of components, distance between previous and next pair of components, width and height of left and right components, along with global components like average height, width of grouped components and average distance between components. Rather than segmenting words some methods segment characters. [15] proposed character segmentation utilizing information as you move from background pixels to foreground pixels in horizontal and vertical directions of the character image. Transition is performed based on traversals in left to right, right to left, top to bottom and bottom to top direction. Whenever a transition is encountered from background to foreground, the ratio between location of the pixel and distance across the image in that direction is calculated. [40] proposed character recognition using a combination of transition and direction features, which they called Modified Direction Feature. For each transition, a pair of location of transition and direction transition was stored, which was used for segmentation. Crossing features are used in character recognition, where the idea is to find the number of times a line intersects a trace of the character. If the line intersects at multiple places then the first and last intersection can be used to describe the shape of the character or symbol.

To summarize, most of the previous work on handwriting recognition involved preprocessing inputs, segmenting characters and extracting hand tuned

features from them. This was followed by classifying the symbol using Artificial Neural Networks (ANN) [12] or Random Forests, and generating the final sequence using Hidden Markov Models (HMM).

Chapter 3

Methodology

For Dysarthric Speech Recognition this thesis explored two different Deep Neural Network architectures:

1. The same pipeline for feature extraction as in GMM-HMM system followed by pre-training of Deep Belief Network (DBN) using Restricted Boltzman Machine (RBM) [23] followed by Framewise training using Deep Neural Network (DNN). We refer this as DNN Framewise training

2. Input a 2D spectrogram to a Convolutional Neural network (CNN) - Recurrent Neural Network (RNN) model followed by Connectionist Temporal Classification (CTC) cost function. We refer this as CNN-RNN-CTC training.

For Offline Handwriting Recognition, we use Multidimensional Recurrent Neural Networks, which takes 2D images as an input as opposed to extracting 1D features beforehand and giving it as an input to a RNN.

3.1 DNN Framewise Training

Figure 3.1 describes a pipeline for feature extraction in a DNN Framewise training system. The following sections describe each component or block

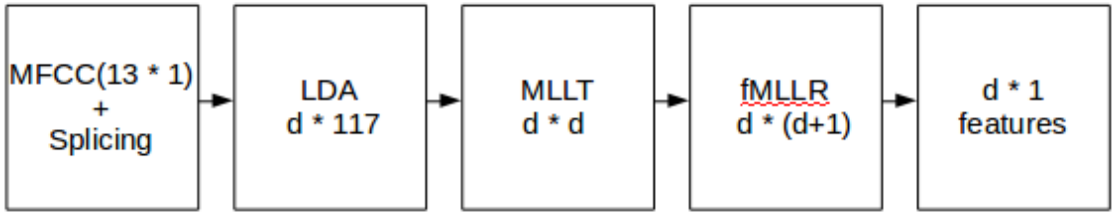

Figure 3.1: Feature extraction pipeline for DNN.

in this pipeline.

3.1.1 Mel-Frequency Cepstral Coefficients

In a typical automatic speech recognition system, the first step is to extract features, i.e identifying components in the audio that represent linguistic content and discarding other unwanted stuff like noise. Generated speech is filtered by the shape of the vocal tract including tongue, teeth, lips etc. This shape determines what sound is generated, which we call phoneme or phones. The shape of the vocal tract manifests itself in the form of short time power spectrum and the job of MFCCs [22] is to accurately represent this short time power spectrum. MFCCs are widely used in speech recognition and the following describes the high level steps on how to extract those features for ASR.

1. For a given speech waveform/signal, splice into short frames.
2. For each short frame, calculate the periodogram estimate of the power

spectrum.

3. Apply a Mel filterbank to the power spectra, then sum the energy in each filter.

4. Take the log of all the filterbank energies.

5. Take the Discrete Cosine Transform (DCT) of log filterbank energies.

6. Only keep first 13 coefficients.

It is also common to calculate delta (velocity) and delta delta features (acceleration), which is appended to original 13 features to get 39 features. While training classifiers like DNNs, splicing basically involves taking left and right frames from the central frame (usually 4 or 5 frames from left and right) to get more contextual information.

3.1.2 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is commonly used as a dimensionality reduction technique in pattern recognition and machine learning applications. The goal of LDA is to project data onto a lower dimensional space with good class separation. The general idea of LDA is similar to PCA except that in PCA we find the axes that maximizes the variances of our data, while in LDA we are interested in finding axes that both maximizes separation between multiple classes and minimizes the spread within each class. Following are the steps for performing LDA.

1. Compute d dimensional mean vectors for different classes in the data set.

2. Compute scatter matrices both within class and in between classes.

3. Calculate eigenvectors and corresponding eigenvalues for the scatter matrices.

4. Sort the eigenvalues in descending order and choose top k eigenvectors corresponding to those eigenvalues. This will be $d \times k$ dimensional matrix W.

5. Use this $d \times k$ dimensional matrix to transform the samples to a new subspace. i.e if $X = n \times d$ and $W = d \times k$, then after performing LDA , we will get $n \times k$ dimensional matrix.

3.1.3 Maximum Likelihood Linear Transformation

Maximum Likelihood Linear Transformation [17] is a feature transformation technique, where a feature transformation matrix is estimated to perform transformation. The objective function reduces average per frame log likelihood of the transformed features given the model. We initially start the model building process using the features transformed using LDA. After running some iterations, we accumulate statistics like mean and covariance to update the MLLT transformation matrix. The transformation matrix obtained is used to transform models mean. This feature transformation is discriminative, where it tries to improve separability of acoustic class in the feature space.

3.1.4 Feature space Maximum Likelihood Linear Regression

Most of the adaptation techniques fall in two main categories, i.e speaker normalization, where input speech is normalized to match speech of speakers on which the model was trained on, and model adaptation techniques, where parameters of the model are adjusted. Model adaptation techniques like MAP estimation update the model parameters, but the amount of adaptation data required is generally large. However, Maximum Likelihood Linear Regression (MLLR) [16] works even with small amounts of adaptation data. The basic idea behind MLLR is that it transforms the model parameters of HMM using a transformation matrix estimated from adaptation data. The method shares similarities with spectral shift transformation [29], which attempts to map data from a new speaker onto a reference speaker .

If the amount of adaptation data is very small, then a global transformation matrix is estimated for all the different states of the HMM. If more data is available, then the transformation matrix is learned for each state of the HMM. The MLLR approach requires initial parameters as HMM parameters from a Speaker Independent (SI) system. It updates model parameters with new adaptation data, however, only mean parameters are updated because it is assumed that the variation in different speaking styles can be characterized by the means.

Consider a case of continuous density HMM system with gaussian output distribution. The probability density function for i_{th} distribution is given by

$$b_i(x) = g(x|\mu_i, \sum_i) = \frac{1}{(2\pi)^{D/2} \det \sum_i^{1/2}} \exp\{\frac{-1}{2}(x-\mu_i)' \sum_i^{-1} (x-\mu_i)\} \tag{3.1}$$

where D is the dimension of observation vector x and $\sum_i$ is the covariance matrix. The adapted mean ${\mu_i}^-$ is obtained by applying a transformation matrix to the extended mean vector ξ_i:

$${\mu_i}^- = W_i \xi_i$$

where $\xi_i = [\omega, \mu_1, \mu_2 ..... \mu_D]'$, ω is the offset term, which is either 1 or 0 for no offset and W_i is $n \times (n+1)$ matrix.Thus, for a new adapted system the probability density function for i_{th} state is given as:

$$b_i(x) = g(x|\mu_i, \sum_i) = \frac{1}{(2\pi)^{D/2} \det \sum_i^{1/2}} \exp\{\frac{-1}{2}(x-W_i\xi_i)' \sum_i^{-1} (x-W_i\xi_i)\} \tag{3.2}$$

Assuming adaptation data $O = x_1, x_2 ..... x_t$, then the total likelihood of a model generating the observation sequence with model parameters λ is given by

$$P(O|\lambda) = \sum_{\theta \forall \Theta} P(O, \theta|\lambda) \tag{3.3}$$

where θ are states and Θ are all possible combination of states.

In order to obtain a transformation matrix, we need to define an auxiliary objective function.

$$Q(\lambda, \lambda^{'}) = constant + \sum_{\theta \forall \Theta} P(O, \theta|\lambda) \log(P(O, \theta|\lambda^{'})) \tag{3.4}$$

where $P(O, \theta|\lambda)$ is the objective function to be maximized. Maximizing the auxiliary function thereby maximizes the objective function. Thus, successively forming a new auxiliary function with new parameters iteratively maximizes the objective function. Since, only W_i is re-estimated, only output distribution $b(i)$ is affected, so the auxiliary function can be written as

$$Q(\lambda, \lambda^{'}) = constant + \sum_{\theta \forall \Theta} \sum_{t=1}^{T} P(O, \theta|\lambda) \log\left(b_{\theta_t}(x_t)\right) \tag{3.5}$$

Suppose gamma is defined as

$$\gamma_i(t) = \frac{1}{P(O|\lambda)} \sum_{\theta \forall \Theta} P(O, \theta_t = i|\lambda) \tag{3.6}$$

where $\gamma_i(t)$ is the posterior probability of all the paths going through state i at time step t.

$$Q(\lambda, \lambda^{'}) = constant + P(O|\lambda) \sum_{j=1}^{S} \sum_{t=1}^{T} \gamma_j(t) \log\left(b_{\theta_t}(x_t)\right) \tag{3.7}$$

Expanding $\log\left(b_{\theta_t}(x_t)\right)$, we finally get the following equation

$$Q(\lambda, \lambda^{'}) = constant - \frac{1}{2} P(O|\lambda) * \sum_{j=1}^{S} \sum_{t=1}^{T} \gamma_j(t)[Dlog(2\pi) + log \det \sum_{j} + h(x_t, j)] \tag{3.8}$$

where $h(x_t, j) = (x - W_j\xi_j)' \sum_j^{-1} (x - W_j\xi_j)$. We can get the transformation matrix W_s by differentiating (3.8) and equating it to zero.

Sometimes estimating such matrices can be complex and might be computationally expensive, so [16] suggested fast adaptation of features rather than model parameters. So, if $x^- = Ax(t) + b = W * x_{ext}$, where W is extended transformation matrix [b A] and $x_{ext} = [1..x_1....x_T]$. As a result (3.8) changes to

$$Q(\lambda, \lambda') = constant - \frac{1}{2} P(O|\lambda) * \sum_{j=1}^{S} \sum_{t=1}^{T} \gamma_j(t) [Dlog(2\pi) - log(\det A)^2 + log \det \sum_j + h(x_t, j)] \tag{3.9}$$

where $h(x_t, j) = (x_t^- - \mu_j)' \sum_j^{-1} (x_t^- - \mu_j)$.

Differentiating (3.9) with respect to A and equating it to zero we get transformation matrix A. This can be used to transform features and do fast speaker adaptation.

Once the fMLLR features are extracted we perform unsupervised training using Deep Belief Networks with Remote Boltzman Machine (RBM) using the training algorithm Contrastive Divergence with 1-step of Markov Chain Monte Carlo sampling (CD-1). The first RBM has Gaussian-Bernoulli units and the following RBMs have Bernoulli-Bernoulli units. One of the things to take care, while training DNNs with large learning rates and thousands of hidden neurons is that there is a high risk of weight explosion. In order to avoid weight explosion, we compare the variance of reconstruction data in a mini-batch (256 frames per batch) with that of the entire training data. If

the variance of batch data $>> 2x$ larger, then the weights are scaled down for that mini-batch. Here we train layer by layer in an unsupervised way.

We use this pretrained DNN consisting of 6 layers obtained from unsupervised training, and randomly initialize the output layer. We then perform frame wise cross entropy training, as described in [67], where frame wise alignments are obtained from Speaker Adaptive Training (SAT) GMM system. In this phase, we train the DNN to classify frames into triphone states. The training is done using stochastic gradient descent (SGD), with sigmoid activation units and output layer is a softmax layer. In order to prevent the network from overfitting, early stopping criteria is used where cross validation data is used to check accuracy of the network.

3.2 CNN-RNN-CTC Training

Amodei et al. [5] proposed a speech recognition system based on CNN + RNN + CTC. The architecture tries to avoid the conventional feature extraction pipeline as describe in section 3.1. Rather than extracting features beforehand, a spectrogram is used as input to the model. Following sections describe each component of the architecture.

3.2.1 Convolutional Neural Network

One of the drawbacks of regular neural networks is that as the dimensions of image increases the number of parameters generally increases. For example, consider an image $32 \times 32 \times 3$ ($width \times height \times channel$) and if

we stretch this image vertically we have input as 3072 pixel values and every neuron in the hidden layer has 3072 parameters. So, if there are 10 hidden neurons in the 1st hidden layer, we will have total 30720 parameters. Now, if we increase the size of an image to $224 \times 224 \times 3$, then total number of parameters would be $150528 \times 10 = 1505280$. Thus, in terms of computation this will take lot of resources and time. CNNs tackle this problem by sharing weights across the 2D input image. In CNNs, we have convolutional filters that share parameters and slide across width and height (taking entire depth), while performing a convolution operation. Thus, if a filter is of size $3 \times 3 \times 3$, then we would do a dot product with input image patch of size $3 \times 3 \times 3$. As we slide the filter across every spatial position, we produce a 2 dimensional activation map. One can intuitively think of these activation maps as filters that are responding, when they see a particular pattern in the input image like edges or orientation or some blotches of color.

It is not practical to have the receptive field of a filter to be the same as the image width and height, because of the concerns related to increase in number of parameters as explained above. The amount of receptive field is a hyper parameter (i.e width and height, and depth is same as the depth of input). We sample the audio signal at a rate of 16KHz with a stride of 10ms and window size of 20ms. 15ms window size was also tried for speakers with speech impairment as described in [44] but it did not perform better than window size of 20ms (word error rate (WER) increased by 2%). In the proposed architecture, the receptive field of the convolutional filter for the first

layer is 41×11 ($height \times width$) and for the second layer is 21×11 as described in [5]. For the 1st layer, we use a stride of 2 both across width and height and for the 2nd layer stride of 2 is used horizontally and 1 vertically.

3.2.2 Recurrent Neural Network

Recurrent Neural Networks (RNN) are like feed forward neural networks with one exception, they allow recurrent connections coming back to the neurons. RNNs have shown to perform good in sequence to sequence learning tasks. One of the major problems with RNNs is the problem of vanishing gradients and exploding gradients [6]. As a result, [19] introduced LSTM cell/unit instead of simple RNN cell/unit. LSTM stands for Long Short Term Memory unit, which has three gates: 1) input gate, 2) forget gate and 3) output gate. These gates help to control the amount of information that should flow through the network, along with a cell state. The Input gate function is to pass or discard input given to the network. The Forget gate function is to decide whether information from previous time steps is important or not. If the information is important then it is passed to cell state of LSTM. The Output gate functionality is to decide whether the output from LSTM unit should be given as an input to the next time step LSTM unit or not. Using LSTM unit in place of vanilla RNNs has shown tremendous improvements in sequence learning tasks. There are also other variations of LSTM units like Gated Recurrent Units (GRU) [8]. GRU's are simpler to understand and implement and have shown to give similar performance as that of LSTMs. This

thesis experimented with different depth of layers and with different hidden units for Recurrent Layers (i.e with RNN, LSTM, GRU units). These experiments show that LSTMs and GRUs give similar accuracy, but with simple RNN unit performance deteriorates for a depth of less than 5 layers. This is because the number of parameters in RNNs are less than that in LSTMs and GRUs per unit/cell. However, if the number of recurrent layers are more than 5, then the performance of all the units is more or less the same. As a result, simple RNN units are used for recurrent layers. One of the advantages of using simple RNN units is that there is less complexity, when it comes to implementing big networks (i.e forward and backward computation) and also the amount of time taken to compute output values for hidden neurons is less. In this architecture, we have used 7 recurrent layers with simple RNN units above the convolutional layers.

3.2.3 Batch Normalization

One of the major problems with very large networks is its difficulty to train, where the gradients might explode or vanish or it might be difficult for the network to converge. This calls for careful initialization of weights such that the variance across each layer remains the same. Ioffe et al. [28] proposed Batch Normalization for training deep neural networks. Training deep neural networks is complicated by the fact that the distribution of each layer's input changes as the parameters from the previous layers change. This results in careful initialization of weights and hard to train models with saturating

non-linearities like sigmoid, tanh. This phenomenon is known as internal covariance shift. Batch Normalization addresses the problem by normalizing the layer's input. The change in distribution of layers causes the network to keep on continuously adapting to a new distribution. This change in distribution is called covariance shift [59].

Fixed distribution of inputs will have positive impact on the layers outside the sub network. For example, for a sigmoid layer $\frac{1}{1+\exp^{-}x}$, where $x = Wu + b$, gradients passing through neurons will be close to zero, if the absolute value of x is very large. The gradients will only pass, when absolute value is less. This impact is amplified if the network is a deep network with many layers stacked one after another. Thus, if the distribution of inputs remain more stable, the optimizer will be less likely to get stuck and would accelerate the training. The idea of batch normalization is similar to whitening. [36] showed that whitened inputs i.e with mean 0 and unit covariance, tend to converge faster. As each layer observes input from previous layer it would be advantageous to have whitening of inputs at each layer. By whitening the inputs, we can achieve fixed distribution of inputs that will remove the ill effects of the internal covariance shift. In order to perform whitening at every layer, within whitening framework we need to calculate the covariance for every layer and also perform complex back propagation. Thus, in order to perform whitening, [28] proposed the following, which is easy to compute and also differentiable. Here we normalize each feature independently to have zero

mean and unit variance. The following is the batch normalization algorithm.

Input: Values of x over mini-batch $B = x_{1...m}$. Parameters to be learned are γ, β

$$Output : Y_i = BN_{\gamma,\beta}(x_i) \tag{3.10}$$

Calculate min-batch mean

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{3.11}$$

Calculate min-batch variance

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \tag{3.12}$$

Normalized input is given by

$$x_i^- = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{3.13}$$

Now scaling and shifting the input, this is done because sometimes the network wants distribution to be non-zero centered or have a different spread.

$$Y_i = \gamma x_i^- + \beta \tag{3.14}$$

Backpropagation through the network is given as follows.

Derivative with respect to output x_i^-

$$\frac{\partial E}{\partial x_i^-} = \frac{\partial E}{\partial Y_i}\gamma \tag{3.15}$$

Derivative with respect to variance

$$\frac{\partial E}{\partial \sigma_B^2} = \sum_{i=1}^{m} \frac{\partial E}{\partial x_i^-}(x_i - \mu_B)\frac{-1}{2}(\sigma_B^2 + \epsilon)^{-3/2} \tag{3.16}$$

Derivative with respect to mean μ_B

$$\frac{\partial E}{\partial \mu_B} = (\sum_{i=1}^{m} \frac{\partial E}{\partial x_i^-}\frac{-1}{\sqrt{\sigma_B^2 + \epsilon}}) + \frac{\partial E}{\partial \sigma_B^2}\frac{\sum_{i=1}^{m} -2(x_i - \mu_B)}{m} \tag{3.17}$$

Derivative with respect to input x_i

$$\frac{\partial E}{\partial x_i} = \frac{\partial E}{\partial x_i^-}\frac{-1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{\partial E}{\partial \sigma_B^2}\frac{2(x_i - \mu_B)}{m} + \frac{1}{m}\frac{\partial E}{\partial \mu_B} \tag{3.18}$$

Derivative with respect to γ

$$\frac{\partial E}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial E}{\partial Y_i} x_i^- \tag{3.19}$$

Derivative with respect to β

$$\frac{\partial E}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial E}{\partial Y_i} \tag{3.20}$$

With batch normalization we do not need to worry as much about initialization of weights and it also helps the network converge faster. We have used

batch normalization after every layer, for recurrent layer we apply batch normalization only to the input coming from the same time step. Suppose W_{ip} is the weight for input layer at time step t and W_h is the weight for hidden layer and b is the bias. Then output O is

$$O_t(i) = W_{ip}x_t(i) + W_h x_{t-1}(i) + b \tag{3.21}$$

with batch normalization applied only to input from time step 't', we have the following equation

$$O_t(i) = BN(W_{ip}x_t(i)) + W_h x_{t-1}(i) + b \tag{3.22}$$

[5] showed the above configuration gave better results.

3.2.4 Dropout

Srivastava et al. [61] proposed Dropout as one of the powerful ways to regularize network to prevent overfitting. The idea behind dropout is that you do not want neurons to be too much dependent on each other and rather be able to independently work towards understanding the features and fire when required. This is done by dropping some of the neurons in every layer with some random probability during every training sample or mini batch. This forces the neuron to learn independently, even when its neighbors are not firing and thereby learning hidden representation in the features. At training

time for every batch in each epoch, we only keep hidden units with probability $p = 0.5$ and drop hidden units with probability $(1 - p)$. At testing time, we need output of neuron to be same as the expected output at training time. So for example, suppose we have an output neuron x. With dropout, the expected output from neuron x will be $px_{prev} + (1 - p)0$, where x_{prev} is 0 for dropped units in previous layer and px_{prev} is an average of number of outputs from previous layer contributed to the output of neuron x. At test time, all neurons will be active and we need to make sure the expected output of neuron $x \rightarrow px_{prev}$ to keep it same as it was during training. As discussed in [61] dropout has an effect of averaging an ensemble of exponential models and thus better generalizes. In our model, we used dropout only in convolutional layers. We tried dropout for recurrent layers, we saw improvement when applied to 1st recurrent layer but when we applied to more than 1 recurrent layer, we did not see much improvement. Also we faced memory issues, because we need to store an additional matrix to keep track of dropped units. So, in our final architecture, we only use dropout with p = 0.4 in convolutional layers. We tried p = 0.5, which is standard but p = 0.4 gave better results.

3.2.5 Connectionist Temporal Classification

Frame wise training of neural networks require separate label or ground truth values for each input frame. As a result, training data needs to be pre-segmented and network only outputs local classification. Any global aspect must be modeled externally (i.e. likelihood of two labels occurring together).

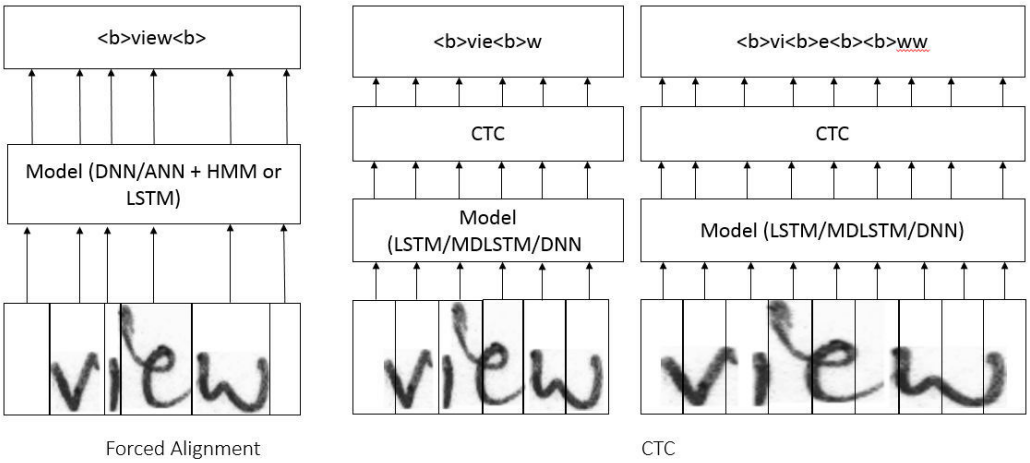

Figure 3.2: Difference Between Forced Alignment and CTC.

This requires further post processing to get a final label sequence. Previous methods used RNNs for temporal classification, then the outputs of RNNs, were given to HMMs, to get final labeling sequence. Figure 3.2 describes the difference between forced alignment (pre-segmented input) and CTC.

In forced alignment, for every input character segment/frame you have a target label, and RNNs calculate loss and update gradients using those target labels. In Connectionist Temporal Classification (CTC), the input sequence can be sliced into many frames as can be seen on the right in Figure 3.2, for the same word. The CTC model takes input i.e. probabilities the network predicts for that slice of an input (frame), then models dependencies between character labels. The final sequence modeling is done using the Forward Back-

ward algorithm used in HMM, except here there are no transition probabilities as you go from one state to another state. We can get posterior probabilities from RNNs, and these are used to model a final label sequence. The Forward Backward algorithm works by finding the best possible labeled sequence, by maximizing the probability of a labeled sequence. The Forward Backward algorithm uses dynamic programming to find the best possible labeled sequence for a given input. The CTC cost function is given as follows.

$$O = -\log(\prod_{(x,z)\in S} p(z|x)) = -\sum_{(x,z)\in S} \log p(z|x) \tag{3.23}$$

where x is the training sample, z is the generated sequence and S is the training data. Because the function is differentiable, the derivative with respect to weights can be calculated with back propagation. In order to calculate gradients with respect to weights, we need to calculate gradients with respect to output first. To calculate gradients with respect to output, we need to find total number of paths going through each label using the forward backward algorithm. Figure 3.3 explains how forward backward algorithm in CTC works.

The white circles represent blank labels, while the filled circles represent labels. In Figure 3.3 the number inside the circle represents total number of paths that go through each circle, i.e for alphas from left to right and for betas from right to left. The total number of paths going through any circle

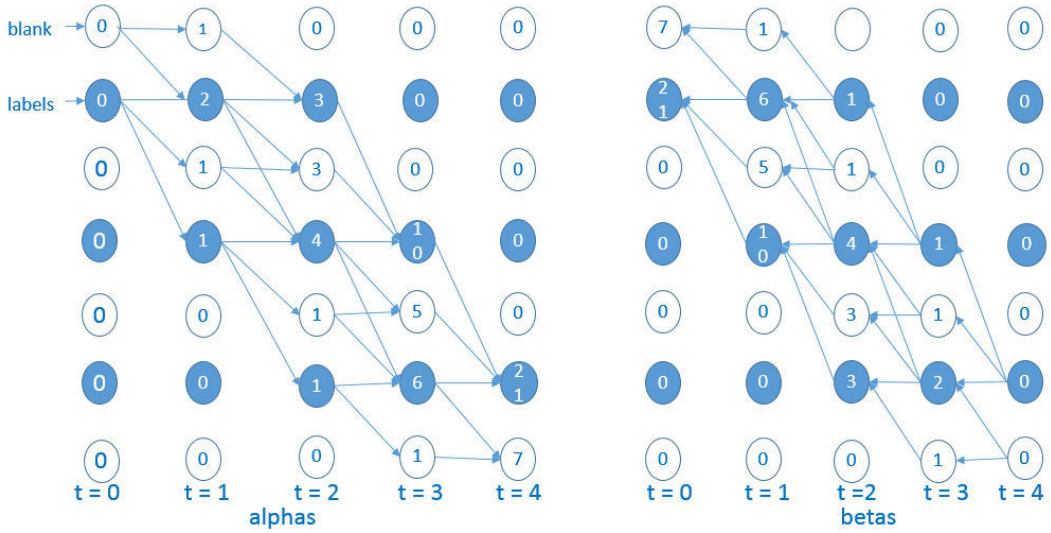

Figure 3.3: Forward Backward Algorithm for CTC.

is given by $\alpha(k) * \beta(k)$, where k represents a label. Now, the gradient with respect to the output is given by

$$\frac{\partial p(z|x)}{\partial y_k^t} = \frac{1}{y_k^t} \sum_{s \in lab(z,k)} \alpha_t(s)\beta_t(s) \tag{3.24}$$

where y_k^t is the output at t time step for label k and summation represents the sum across the same label, i.e if the ground truth word is KITKAT, then differentiating with respect to 'K', we sum across 'K' twice. When backpropagating the gradients, we need to find gradients with respect to the output just before the activation is applied.

$$\frac{\partial O}{\partial a_k^t} = -\sum_{k'} \frac{\partial O}{\partial y_{k'}^t} \frac{\partial y_{k'}^t}{\partial a_k^t} \tag{3.25}$$

Here k' refers to all the labels and k is the k^{th} label. The following equation gives gradients with respect to output before the activation is applied.

$$\frac{\partial O}{\partial a_k^t} = y_k^t - \frac{1}{p(z|x)} \sum_{s \in lab(z,k)} \alpha_t(s) * \beta_t(s) \tag{3.26}$$

Once gradients with respect to output before the activation are calculated, gradients with respect to weights can be calculated using the chain rule.

3.2.6 Decoding

During test time, the network predicts posterior probabilities for each character at each slice or frame as shown in Figure 3.2. Basic decoding involves choosing the charcter with maximum probability and then removing blanks and repeated characters from the final sequence. For example, if network outputs a word $NEEE - U - -RR - ALL$, where '-' represents 'blank', then after removing repeated characters and blanks the final output will be $NEURAL$. However, it would be better to get contextual information, while decoding the final output sequence, i.e how likely is $P(U|NE)$ or how likely is $P(A|NEUR)$. In short, it would be great to have a n-gram language model that helps in decoding the output sequence taking previous contextual information into consideration. [41] proposed beam search decoding using a

character language model with CTC. Figure 3.4 describes beam search decoding with CTC as in [41]. Given the likelihood from our model and our character language model for each time step t and for each string s in our previous hypothesis set Z_{t-1}, we consider extending s with a new character. Blanks and repeat characters are handled separately, while for every other character extension we use character language model when computing s. Notation Z_0 is an empty string $\emptyset$, $\varsigma^{'}$ is character set excluding blank, $s + c$ is concatenation of character 'c' with string s, $|s|$ is length of string s, $P_b(c|x_{1:t})$ and $P_{nb}(c|x_{1:t})$ is probability of character ending and not ending with blank conditioned on input upto time step 't' and $P_{tot}(c|x_{1:t}) = P_b(c|x_{1:t}) + P_{nb}(c|x_{1:t})$

3.2.7 Learning Hidden Unit Contributions

Swietojanski et al. [65] proposed learning hidden unit contributions for Deep Neural Networks (DNNs). An additional amplitude parameter is defined for each hidden unit and the amplitude parameters are learned for each speaker. The amplitude parameter is learned on some amount of adaptation data for speaker m. We modify speaker independent model by learning $\theta_m^l = r_1.....r_m$ for l^{th} hidden layer. [65] learned θ_m for fully connected layers but we only have a fully connected layer as the final layer, so we learn θ_m after the recurrent layer. One of the reasons we did not learn the hidden unit contribution for the final layer was because [65] saw adapting bottom layers resulted in better performance than adapting top layers. Thus, we adapt 1st

Inputs CTC likelihoods $p_{\text{ctc}}(c|x_t)$, character language model $p_{\text{clm}}(c|s)$
Parameters language model weight α, insertion bonus β, beam width k
Initialize $Z_0 \leftarrow \{\varnothing\}$, $p_{\text{b}}(\varnothing|x_{1:0}) \leftarrow 1$, $p_{\text{nb}}(\varnothing|x_{1:0}) \leftarrow 0$
for $t = 1, \ldots, T$ **do**
  $Z_t \leftarrow \{\}$
  for s **in** Z_{t-1} **do**
    $p_{\text{b}}(s|x_{1:t}) \leftarrow p_{\text{ctc}}(\text{-}|x_t)p_{\text{tot}}(s|x_{1:t-1})$ ▷ Handle blanks
    $p_{\text{nb}}(s|x_{1:t}) \leftarrow p_{\text{ctc}}(c|x_t)p_{\text{nb}}(s|x_{1:t-1})$ ▷ Handle repeat character collapsing
    Add s to Z_t
    for c **in** ζ' **do**
      $s^+ \leftarrow s + c$
      if $c \neq s_{t-1}$ **then**
        $p_{\text{nb}}(s^+|x_{1:t}) \leftarrow p_{\text{ctc}}(c|x_t)p_{\text{clm}}(c|s)^\alpha p_{\text{tot}}(c|x_{1:t-1})$
      else
        $p_{\text{nb}}(s^+|x_{1:t}) \leftarrow p_{\text{ctc}}(c|x_t)p_{\text{clm}}(c|s)^\alpha p_{\text{b}}(c|x_{1:t-1})$ ▷ Repeat characters have "-" between
      end if
      Add s^+ to Z_t
    end for
  end for
  $Z_t \leftarrow k$ most probable s by $p_{\text{tot}}(s|x_{1:t})|s|^\beta$ in Z_t ▷ Apply beam
end for
Return $\arg\max_{s \in Z_t} p_{\text{tot}}(s|x_{1:T})|s|^\beta$

Figure 3.4: Beam Search Decoding with CTC [41].

recurrent layer. We tried adapting more than one recurrent layer but we saw the performance of the network worsened. If W^l_{ip} is input layer weight, W^l_h hidden layer weight and b^l is bias for l^{th} layer, then after adaptation or learning amplitude parameter, we get (3.27), where $\odot$ is an element wise operation.

$$O_t = a(\theta^l_m) \odot \phi(BN(W^l_{ip}x_t) + W^l_h x_{t-1} + b^l) \tag{3.27}$$

where ϕ represents any non-linear activation function like sigmoid or ReLU. $a(.)$ is chosen in a way to constrain values of r to [0,2], we tried range [0,4] but we got same accuracy. We use $a(c) = max(0, min(c, 2))$, other options are $a(c) = \frac{2}{1+\exp(-c)}$. Thus, learning speaker dependent (SD) parameters helps to reduce WER by giving more importance to specific hidden units and less to others. One of the advantages of using LHU is that the number of learned parameters equals $\sum_{i=1}^{n} h_n$, where h_n is number of hidden units in the l^{th} layer. We used the same approach as described by [65] but we used it for recurrent layers and observed reduced WER without changing any other parameters. However, we also saw further reduction in WER, when the LHU layer was learned after adapting model on 25% testing data.

3.3 Multidimensional Recurrent Neural Network

Recurrent Neural Networks (RNNs) traditionally take 1D features as input. For a 2D image, we need to extract features using traditional methods or CNNs. Extracted 1D features are fed as an input to a RNN with RNN

or LSTM unit/cell. CNNs are local in nature i.e. the features extracted are local to the neighboring pixels. As a result, a lot of global information is lost, which otherwise would have helped the network learn interesting patterns in an input image. In order to get global information, we can directly use a 2D image as an input to Multidimensional Recurrent Neural Networks (MDLSTM or MDRNN) as described in [58].

The input image is traversed from the directions left, right, top and bottom, so every pixel has a global information from other pixels in an image, along with local information. MDLSTM stores or forgets this global information using a set of forget gates for each dimension. As a result, with this architecture, an input of D dimensions can be used, with D forget gates storing or discarding information from each of the D dimensions. RNNs can use information from left and right direction by using Bidirectional Recurrent Neural Networks [58]. Using BRNNs has shown improvement in many sequence to sequence learning tasks. As a result, this research uses BRNNs to learn from different directions of an image i.e. left, right, top and bottom. MDLSTM/MDRNNs are one of the main components of this architecture because it is at this stage features from an image are learnt by stacking many layers of MDLSTM, forming a deep architecture. Mathematically the forward pass of the model is represented as follows.

$$i_t = f_1(W_i \cdot a_t + \sum_d (H_i^d \cdot h_{t-1}^d + C_i^d \cdot c_{t-1}^d) + b_i) \tag{3.28}$$

$$f_t^{d'} = f_1(W_f \cdot a_t + H_f^{d'} \cdot h_{t-1}^{d'}) + C_f^{d'} \cdot c_{t-1}^{d'} + b_f^{d'} \tag{3.29}$$

$$c_t^* = f_2(W_{c_t^*} \cdot a_t + \sum_d (H_{c_t^*}^d \cdot h_{t-1}^d) + b_{c_t^*}) \tag{3.30}$$

$$c_t = \sum_d (f_t^d \odot c_{t-1}^d) + i_t \odot c_t^* \tag{3.31}$$

$$o_t = f_1(W_o \cdot a_t + \sum_d (H_o^d \cdot h_{t-1}^d) + C_o \cdot c_t + b_o) \tag{3.32}$$

$$h_t = o_t \odot f_2(c_t) \tag{3.33}$$

where

a_t : input

i_t : input gate

f_t : forget gate

o_t : output gate

c_t : cell state

c_t^* : output from tanh activation

h_t^d : hidden state at 't' time step from d dimension

$C_{i,o,f}$: peep hole weights for input, forget and output gate

f_1 : sigmoid activation

f_2 : tanh activation

W_i : input gate weights

W_f : forget gate weights

W_o : output gate weights

$W_{c_t^*}$: input weight

H_i : hidden state weight for input gate

H_f : hidden state weight for forget gate

H_o : hidden state weight for output gate

$H_{c_t^*}$: hidden state weight for input

b_i : bias for input gate

b_f : bias for forget gate

b_o : bias for output gate

$b_{c_t^*}$: bias for input

$\odot$: element wise

Figure 3.5 shows the architecture of a MDLSTM unit/cell. In order to better understand the architecture of a MDLTM unit, we only draw all the gates in the middle unit (i.e current position, which takes input from 't' time step), while other two are units in X and Y direction from 't-1' timestep. Blue circles in the middle unit are the gates with sigmoid activation. On the lower left is an input gate and on the upper left is an output gate. On the right are two forget

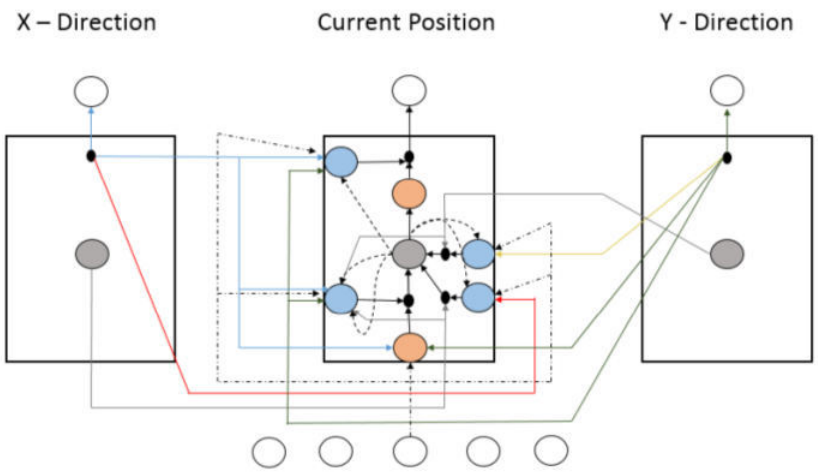

Figure 3.5: MDLSTM unit architecture.

gates taking input from X direction (left unit with output shown in red arrow) and Y direction (right unit with output shown in yellow arrow). Grey circles are cell states of each MDLSTM unit and the output of cell states from each direction goes to the specific forget gate. Orange circles are tanh activation functions. The dotted lines inside the unit from the cell state (grey color) are peep hole connections and dotted lines from the white circle is the input from that particular 't' timestep. Along with MDRNN/MDLSTM, the network has a CTC cost function, which is covered in Section 3.2.5, and decoding stage which is covered in Section 3.2.6.

3.4 Dataset

3.4.1 Dysarthric Speech Recognition

The TORGO database [56] is the only freely available database for continuous dysarthric speech recognition. The database consists of 15 subjects of which eight (five males, three females) are dysarthric and seven (four males, three females) are control (non-dysarthric) speakers. All the dysarthric participants have been diagnosed by a speech-language pathologist according to Frenchay Dysarthria Assessment [11], which is an assessment to evaluate overall clinical intelligibility and the motor functions of the articulators. Based on the assessment, four subjects F01, M01, M02 and M04 are severely dysarthric. One participant, M05 is moderately to severely dysarthric and one subject F03 is moderately dysarthric. The other two subjects have very mild dysarthria. Total dysarthric speech data is 5.45 hours and non-dysarthric speech data is 8.18 hours, which includes speech recorded from both microphone and headphone. The dataset contains non-words, short words, restricted sentences and unrestricted sentences. Non-words include repetition of sequences like /iy-p-ah/ or /p-ah-t-ah-k-ah/, high-pitch and low-pitch vowels. Short words include repetition of English digits like 'yes' or 'no', 410 words from the word intelligibility section of the Frenchay Dysarthria Assessment [11] and the Yorkston-Beukelman Assessment of Intelligibility of Dysarthric Speech [74], phonetically contrasting words and common words from the British National Corpus. Restricted sentences include preselected phoneme rich sentences like 'She had your dark suit in greasy wash water all year', sentences from The Grandfather

passage, 622 sentences from intelligibility section of the Yorkston-Beukelman Assessment of Intelligibility of Dysarthric Speech [74] and MOCHA-TIMIT database [71]. Unrestricted sentences involved asking participants to spontaneously describe 30 images of interesting situations taken randomly from the cards in the Webber Photo Cards: Story Starters collection [68].

From the above categories we remove 398 speech samples from our entire data (which includes control and dysarthric speech data) because we do not have proper reference transcription for these samples. These samples are mainly from Non-words and Unrestricted sentences section. A speech sample represents an entire audio file, i.e it can be an audio file of just single word spoken by speakers or sentences spoken by speakers. As we can see in Table 3.1 and Table 3.2, row Min words represents speech sample having just single word as ground truth, Max words represents speech sample that has maximum number of words, Total words represents all the words, which are ground truths for speech samples, Min time represents the shortest speech sample in seconds, Max time is longest speech sample in seconds, Total time is the sum of duration of all speech samples in seconds, Single word samples are speech samples with single spoken word, Multi word samples represents sentences spoken by speakers and Total samples represents total amount of speech samples for each speaker.

Figures from 3.5 (a)-(i) show spectrogram of dysarthric speakers and one for control male speaker MC02 saying word *trouble*. In Figure 3.5 (a) top part represents spectrogram (X-axis time and Y-axis frequency), where

Speaker	F01	M01	M02	M04	M05	F03	F04	M03
Degree	Severe	Severe	Severe	Severe	Severe-Mild	Mild	Mild	Mild
Min words	1	1	1	1	1	1	1	1
Max words	14	14	14	14	14	14	14	14
Total words	556	1849	1898	1661	1505	2794	1708	2022
Min time (secs)	0.75	1.00	0.15	0.31	0.008	0.15	0.6	0.31
Max time (secs)	12.6	179.94	17.7	23.25	20.85	193.82	10.35	13.05
Total time (secs)	560	2928	2826	2859	2687	3028	2221	2542
Single word samples	188	561	588	508	469	820	506	616
Multi word samples	40	178	184	151	141	277	169	190
Total samples	228	739	772	659	610	1097	675	806

Table 3.1: Statistics of Dysarthric Speakers data

Speaker	FC01	FC02	FC03	MC01	MC02	MC03	MC04
Degree	None	None	None	None	None	None	None
Min words	1	1	1	1	1	1	1
Max words	14	15	15	15	15	15	14
Total Words	674	5524	4996	5478	2847	4183	4091
Min time (secs)	2.25	0.31	0.31	0.60	1.95	0.15	0.3
Max time (secs)	7.35	7.5	10.05	9.0	8.55	6.75	6.45
Total time (secs)	1067	6195	5024	6470	3942	3517	3237
Single word samples	244	1608	1402	1572	830	1239	1195
Multi word samples	52	575	522	569	282	422	419
Total samples	296	2183	1924	1241	1112	1661	1614

Table 3.2: Statistics of Control Speakers data

dark region represents amount of energy present at particular frequencies and bottom represents phones for that particular section or frame of waveform. We can get a visual understanding of how different is the pattern across dysarthric speakers and also how difficult the task is to recognize a given word.

3.4.2 Handwriting Recognition

For handwriting recognition, the IAM Handwriting database, which is well known database across researchers in handwriting recognition community is used. The database was first published in International Conference on Document Analysis and Recognition [42]. The database contains forms of unconstrained handwritten text, which were scanned at a resolution of 300dpi and saved as PNG images with 256 gray levels. Figure 3.6 shows some samples from the database. Following are the details of the dataset.

- 657 writers contributed samples of their handwriting
- 1539 pages of scanned text
- 5685 isolated and labeled sentences
- 13353 isolated and labeled text lines
- 115320 isolated and labeled words

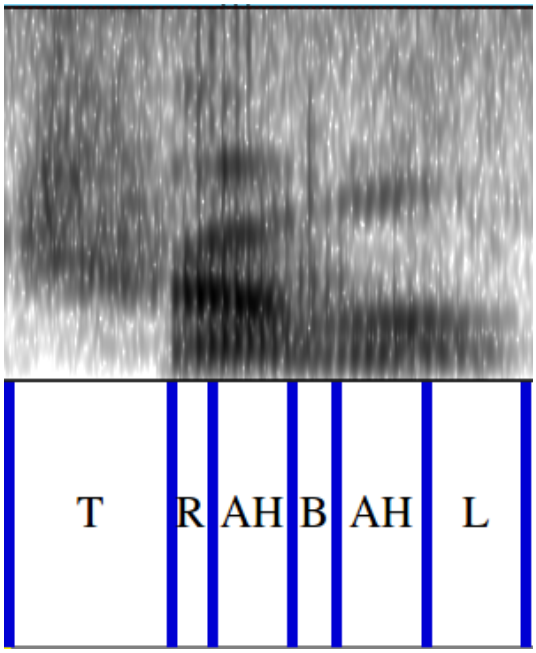

Figure 3.5 (a): Control speaker MC02 saying 'trouble'.

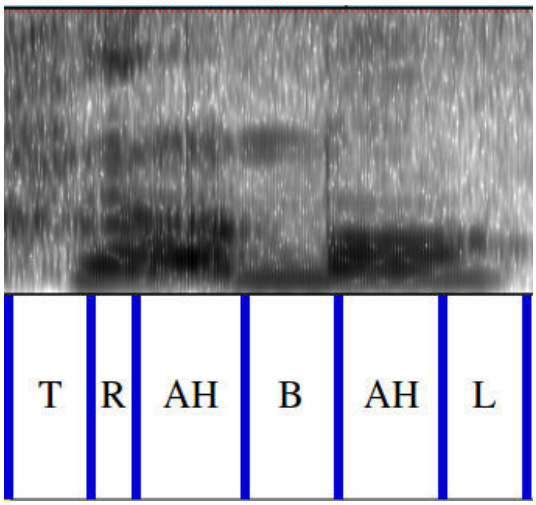

Figure 3.5 (b): F01 saying 'trouble'.

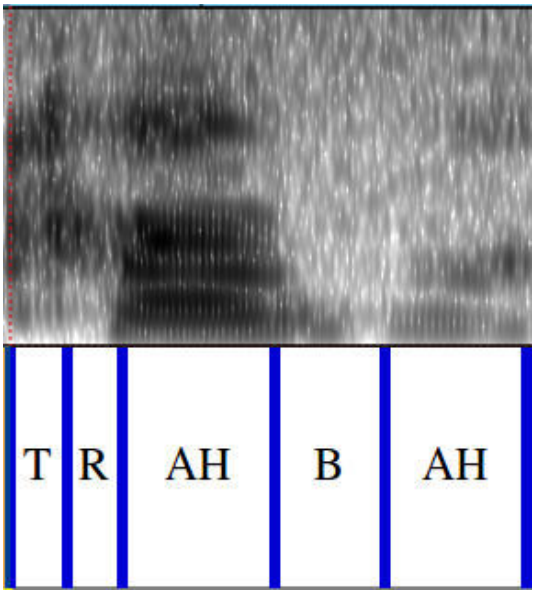

Figure 3.5 (c): F03 saying 'trouble'.

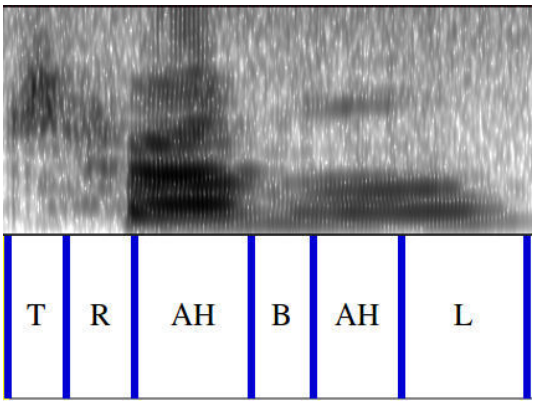

Figure 3.5 (d): F04 saying 'trouble'.

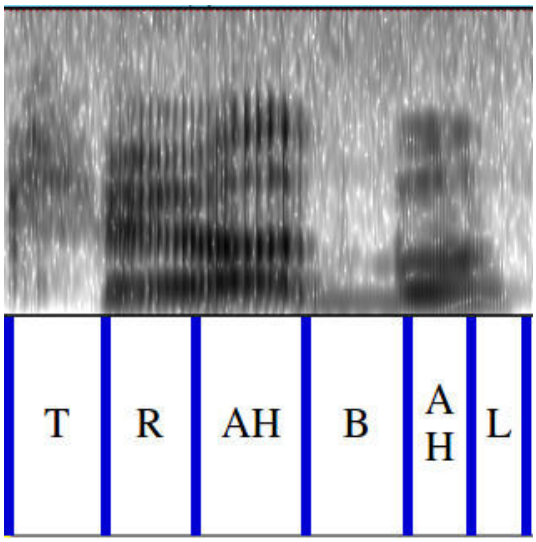

Figure 3.5 (e): M01 saying 'trouble'.

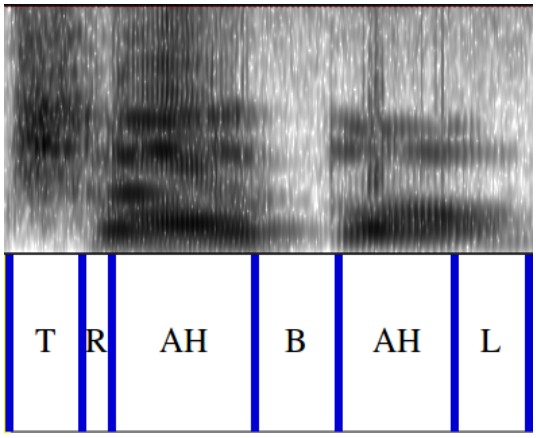

Figure 3.5 (f): M02 saying 'trouble'.

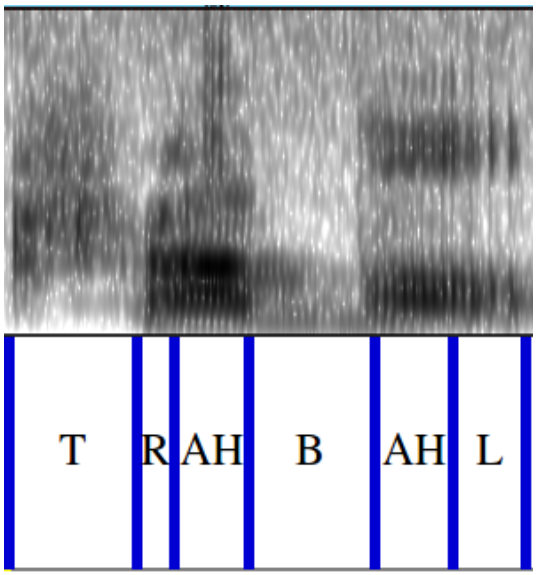

Figure 3.5 (g): M03 saying 'trouble'.

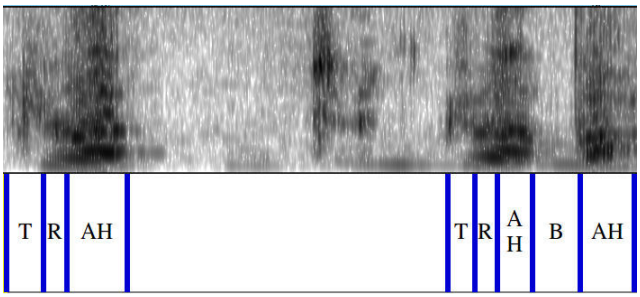

Figure 3.5 (h): M04 saying 'trouble'.

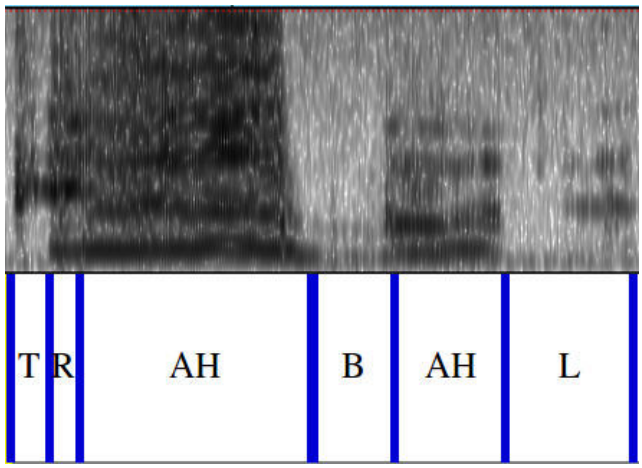

Figure 3.5 (i): M05 saying 'trouble'.

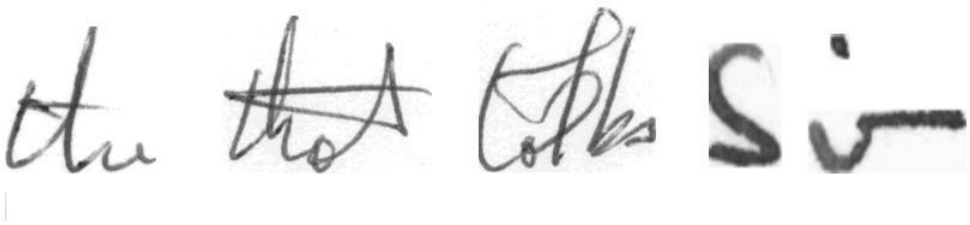

Figure 3.6: IAM Handwriting Recognition Samples.

Chapter 4

Results and Discussion

Deep Neural Networks often rely on there being a large amount of labeled training data available to train these big models. The TORGO database is not a large database, so we adopted the concept of pretraining, which is widely used in computer vision community. In fact, pretraining is not at all a new concept and has been used in [37] for audio classification, where layer by layer RBM's are trained on unlabeled data. We can think of pretraining as providing a better initialization point for weights to start training rather than starting with random initialization of weights. For the CNN-RNN-CTC model, we get the pretrained model from random weight initialization and training the model on the Librispeech database [47]. The Librispeech database has 1000 hours of clear speech data from audio books. We train the model till it converges, the final accuracy in terms of word error rate (WER) is 12 % on Librispeech test-clean dataset, while [5] with similar architecture and using a language model report 5.33 % WER. WER is calculated as

$$\text{WER} = (\text{I} + \text{S} + \text{D})/\text{N}$$

where I - Number of insertions, S - Number of substitutions, D - Number of deletions and N - Total number of words. For the DNN Framewise model, we

perform pretraining in an unsupervised way using fMLLR features as input as described in section 3.1. We do not use the Librispeech 1000 hours of data to pretrain the DNN Framewise model. However, the TORGO database [56] is used for pretraining the DNN Framewise model.

If we use the pretrained model trained using Librispeech data to test dysarthric speakers we get high WER rates and one of the reason for that is the distribution of data is different (i.e. dysarthric speakers tend to have more pauses in their speech and duration of words spoken is longer compared to control speakers). So, we adapt our model to dysarthric speakers by tuning weights of the network and that is done by training our model on both control speakers and dysarthric speakers keeping the test speaker aside. One of the reasons, we do not train on speech data of just dysarthric speakers is because of inter speaker variability across speakers, resulting in the model performing very poorly on test data. Also, we experimented with training only on control speakers data and testing on couple of dysarthric speakers, but that did not perform as good as the model trained on control and dysarthic speakers data. The reason new model (i.e. control + dysarthric speakers) performed better is because the convolutional filters learned to extract better features to capture variation in dysarthric speech and the train set contains same words and sentences spoken by test speaker, so the conditional probability of a character given previous characters is better represented in this model than the previous model trained only on control speakers. Similar behavior was observed by [44]. They also observed deviation in the utterances of six dysarthric subjects and

grouped them in following categories

- Final consonant deletion: Deletion of final consonants that require more articulatory control . For example $feed \rightarrow [f \quad iy]$, $\quad read \rightarrow [r \quad iy]$, it was observed in utterances of F01, M01 and M04.

- Consonant cluster reduction: Omission of consonants in consonant cluster. For example $bright \rightarrow [b \quad ay \quad t]$, $\quad grow \rightarrow [g \quad ow]$, it was observed in utterances of F01, M04 and M05.

- Initial /s/ deletion: When s is followed by a stop word, it is omitted, generally by F01, M01 and M04. For example $spark \rightarrow [p \quad ae \quad r \quad k] \quad spit \rightarrow [p \quad ih \quad t]$.

- Initial /h/ deletion: When a word starts with h , it is omitted, it was observed in M01 . For example $hair \rightarrow [eh \quad r]$, $\quad house \rightarrow [aw \quad z]$.

- Devoicing : Voiceless counterpart of voiced target is produced in the utterances of F03 and M02. For example $deer \rightarrow [t \quad ih \quad r]$, $\quad ride \rightarrow [r \quad ay \quad t]$.

- Prevocalic voicing: Voiceless consonants are voiced for F01, M01 and M04. For example $toe \rightarrow [d \quad ow]$, $\quad feet \rightarrow [v \quad iy \quad t]$.

- Vocalization: When occurred in the end of a word, liquids $|l|$ and $|r|$ are produced as vowels. For example $trouble \rightarrow [t \quad r \quad ey \quad b \quad ow]$, $\quad better \rightarrow [b \quad eh \quad r \quad ah]$. Observed in F01, M02 and M04.

- Stopping: Substitution of fricative with stop consonant, observed in F01, F03, M01 and M04. For example $farm \rightarrow [p \quad aa \quad r \quad m]$.

- Insertion of a short vowel in consonant cluster for speakers M01 and M02. For example $slip \rightarrow [s \quad ih \quad l \quad ih \quad p], \quad blow \rightarrow [b \quad ih \quad l \quad ow]$.

We tried different training strategies like fine tuning only the last layer or fine tuning only RNN layers or only CNN layers just to capture the different speaking style and speaking rate of speakers. In many speakers, a combination of these patterns listed above are observed, as a result we had to train all the layers including convolutional layers, which were good at extracting features from normal speech. While training, we keep held out cross validation data and stop the training if the decrease in WER between two successive epochs is less than 0.1 %. Table 4.1 displays word error rate (WER) of CNN-RNN-CTC training, where the model is trained on all other speakers and 25 % data of test speaker.

Methods	F01	M01	M02	M04	M05	F03	M03	F04
CNN-RNN-CTC	71	88	86	77	90	71	13	24
CNN-RNN-CTC + data aug	73	81	79	74	86	56	13	18
CNN-RNN-CTC + data aug + LHU	71	76	77	72	93	58	12	20

Table 4.1: CNN-RNN-CTC results in % WER.

Even though the model was trained on clean speech data and with (Dysarthric + Control) speakers data, the WER is still high. For example, if the reference transcription is *'he is definitely a notch above us'*, the predicted output by the model is *'dsiiually notsch above mus'*. There is lot of variation

across speakers and to capture that variation by convolutional filters, we can augment the training data, we can learn adaptation parameters for the network by using a subset of test speaker's data, or we can do both. In order to augment the data, we used temporal perturbation and speed perturbation as described in [34]. [32] investigated speech rate modification, where tempo of the signal is modified keeping pitch and spectral envelopes the same. Two copies of data are created by modifying the tempo to 90% and 110% of the original signal. In speed based perturbation, we just resample the signal. Two copies are created by modifying the speed to 90% and 110% of the original rate. Also, there has been research on improving voice recognition of people with parkinson's disease by amplifying the speech sound, so we amplified the speech sound by a factor of 2. We used SoX [1] to make all modifications to the original speech sound. In Table 4.1, row CNN-RNN-CTC + data aug, shows WER after data augmentation.

Since it is difficult to train the model on different ways in which people speak a given word, most of the speech recognition system uses speaker adaptation techniques as described in background section. Though fine tuning the layers can be thought as adaptation to the test speaker, it results in catastrophic forgetting in connectionist networks [13], where previously learned information is erased with new information. This means as we adjust the weights or learn weights that do better on a new speaker, the information learned by the network previously is lost. Thus, in order to avoid that, different speaker adaptation methods are proposed. We tried speaker codes as described in [3],

where once the model is trained, additional adaptation network and speaker code layer is randomly initialized and trained on the development set. Once the adaptation network is trained, we take small amounts of samples from the test data and keep the original network and adaptation network the same, only learning speaker codes. We implemented this method and it did not help much to reduce WER. We observed that backpropagated derivatives were very small to make any changes in randomly initialized speaker codes. Also, we need to store speaker codes for every speaker, which is a cumbersome task.

Another method which has worked well is LHU [65], where they learn hidden unit contributions in DNNs. Since, our architecture only contains a fully connected layer as the final layer, we implemented a LHU layer after the 1st recurrent layer. We tried different combinations of adding LHU layer i.e after 1st RNN layer or after all RNN layers, but adding after the 1st RNN layer gave good performance. One of the reasons, why adding more LHU layers did not help is because every RNN layer has 1760 hidden units. If we add LHU layer after every RNN layer, then we will have 1760 * 7 = 12320 parameters to learn. If we have lot of data for adaptation, then adding more LHU layers will help, however, we only use 25 % of test data, which varies across speakers. Table 4.1 row (CNN-RNN-CTC + data aug + LHU) displays WER after adding a LHU layer.

Based on the results from Table 4.1 we can conclude that augmenting data helps the most, except for M03, which has very mild dysarthria and has intelligibility within ranks of non-dysarthric speakers [44]. Probably modifying

the tempo and speed by 95% and 105% of the original signal might help more than modifying by 90% and 110%, which we used for augmentation. Also, adding LHU adaptation helps for severely dysarthric speakers and not for severely-mild or mild dysarthric speakers as the network tends to overfit for these speakers.

Another way of adaptation is by transforming features as described in Section 3.1. Here we tried different variations of features and tested them on both GMM-HMM methods and DNN-HMM methods. We used Kaldi [52] speech recognition toolkit recipes to extract features and train the GMM-HMM and DNN-HMM systems. Table 4.2 shows WER for all dysarthric speakers in different experiment settings.

Methods	F01	M01	M02	M04	M05	F03	M03	F04
Rudzicz [44]	42	28	25	65	24	19	NA	NA
Mono	74.82	86.70	78.87	87.30	90.50	58.30	40.31	27.52
TRI (tri2a)	73.92	94.21	86.25	90.01	115.33	42.95	23.44	18.50
TRI (tri2b)	78.96	94.16	90.99	91.39	121.24	53.01	25.96	18.38
TRI (tri3b)	54.86	95.13	88.41	88.38	139.79	39.62	11.38	16.39
sGMM(sgmm4a)	56.47	107.57	92.83	91.41	147.72	40.66	8.90	12.59
sGMM2	56.47	107.46	91.15	91.81	146.30	40.41	8.80	12.53
DNN	49.10	85.02	83.61	101.99	119.9	35.43	9.94	12.88

Table 4.2: GMM-HMM and DNN-HMM results in % WER.

Mono: Monophone model trained on MFCC features.

TRI (tri2a): Here we use MFCC features along with velocity and acceleration (i.e delta and delta delta) features to train triphone model.

TRI (tri2b): In addition to extracting MFCCs, we further process MFCC features i.e we perform Linear Discriminant Analysis (LDA) and Maximum

Likelihood Linear Transformation (MLLT) to decorrelate features .

TRI (tri3b): Here we apply fMLLR transformation on top of (MFCC + LDA + MLLT) features. We learn transformation matrix for every speaker and use this transformation matrix to adapt test data to speaker independent system.

sGMM (sgmm4a): We use features from tri3b for training and testing subspace gaussian model.

sGMM2: We do additional fMLLR transformation on speaker dependent (tri3b) features and train and test on subspace gaussian model.

DNN: We follow DNN training procedure listed in section 3.1, with (MFCC + LDA + MLLT + fMLLR) i.e (tri3b features) as input features to our DNN model.

Table 4.2 shows WER for different non-adaptation methods like Mono, TRI (tri2a) and TRI (tri2b) and adaptation methods like TRI (3b), sGMM, sGMM2 and DNN. One of the reasons for reduced WER for [44] is that they created a dictionary based on speaking style of each dysarthric speaker. The reason they created speaker specific dictionaries was because they were working on assistive applications, where they had access to speaker specific information. For each speaker, they analyzed their articulatory characteristics and created a dictionary. For example, speakers F01, M01 and M04 had tendency to delete final consonants, i.e $feed \rightarrow [f \quad iy]$, this fee is replaced by $feed$ from the dictionary. However, in real world it is not possible to create dictionary for each speaker based on their articulatory characteristics.

Based on the results from Table 4.2, we can see feature based adap-

tation methods work (i.e using fMLLR features) better for female speakers than for male speakers. If we compare results from Table 4.1 which performs adaptation at a parameter level and Table 4.2 which performs adaptation at a feature level, we can see the CNN-RNN-CTC model i.e last row of Table 4.1 average WER across dysarthric speakers is 74.5%, while for DNN Framewise model i.e last row in Table 4.2 is 79.17%. However, the DNN Framewise model does not use any data augmentation, so it is not completely a fair comparison. Also, one thing to note is that for CNN-RNN-CTC, the outputs are characters, while for DNN Framewise model, the outputs are context dependent triphone states. Using phones instead of characters as output targets might improve the recognition accuracy of CNN-RNN-CTC model. The WERs also explain the varying characteristics of speakers, where every dysarthric speaker is unique and we cannot just have a generalized model across all the dysarthric speakers.

For Handwriting recognition, we used Multidimensional Recurrent Neural Networks with LSTM cells (MDLSTM) as described in Section 3.3. Post processing is common in handwriting recognition and speech recognition tasks, where incorrect outputs are corrected by finding a proper match in a dictionary or by using a language model. One of the problems with dictionary lookup is that as the size of dictionary increases the amount of time taken to find a perfect match increases. We tried three dictionary sizes, MDLSTM + Vocab (9.5k train dictionary), where only words occurring in training set were used to create dictionary. In the second one, we created a dictionary from web crawl data (around 4GB data), which was used by [7] for training 1 billion word

language model. We did preprocessing to remove stop words and non-ASCII characters, then we take the 50k most frequent words. It took around 10 hours to process the entire test set parallel across 16 cores and we did not see any improvement in Character Error Rate (CER). Table 4.3 row MDLSTM + Vocab (50k dictionary) displays CER. The reason behind no improvement is that there are still 3.6k unique words in the test set, which are not there in the dictionary. Finally, we created a dictionary using all words in IAM dataset and ran dictionary look up. We can see in Table 4.3 row MDLSTM + Vocab (11k both train-test dictionary) that the CER reduced by 1.25 % absolute than the baseline MDLSTM model. In IAM database, the train, val and test split is designed in a way that no writer is repeated in any of the splits. In order to understand more about the network, we designed our own train, val and test split, where we randomized the data and used 80k samples to train, 15k samples to cross validate and 20k samples to test. Table 4.3 row MDLSTM (train 80k-test 20k) displays CER, when trained on 80k samples and row MDLSTM (train 80k-test 20k) + Vocab (11k both train-test dictionary) displays CER after running dictionary lookup. From the CER results we can conclude that the network does much better when it has some understanding of the flow and structure of handwritten characters. Table 4.3 also shows CER for different models trained and tested on IAM dataset. One of the reasons why others get low CER is because they use a language model as a post processing technique to improve upon existing CER, in some cases it reduces CER by 6-7%. [50] created a dictionary of frequent 50k words and a 3-gram language model from

Methods	% CER
pham [50]	5.1
kozielski [35]	5.1
espana [12]	9.8
MDLSTM (Baseline)	26.34
MDLSTM + Vocab (9.5k train dictionary)	29.46
MDLSTM + Vocab (50k dictionary)	26.33
MDLSTM + Vocab (11k both train-test dictionary)	25.09
MDLSTM (train 80k-test 20k)	15.00
MDLSTM (train 80k-test 20k) + Vocab (11k both train-test dictionary)	12.60

Table 4.3: % CER on Test set.

LOB, Brown and Wellington corpora. Some of the passages in IAM test set are from LOB corpora, thus they have more relevant words in the dictionary than the dictionary we created from web crawl data, which helps them to reduce CER but not applicable in real word scenarios.

Figure 4.2 shows some of the filters learned by MDLSTM network in initial layers to better visualize what the network is learning. We can see that some filters are learning to recognize horizontal and vertical edges, while some are separating and highlighting characters from the background.

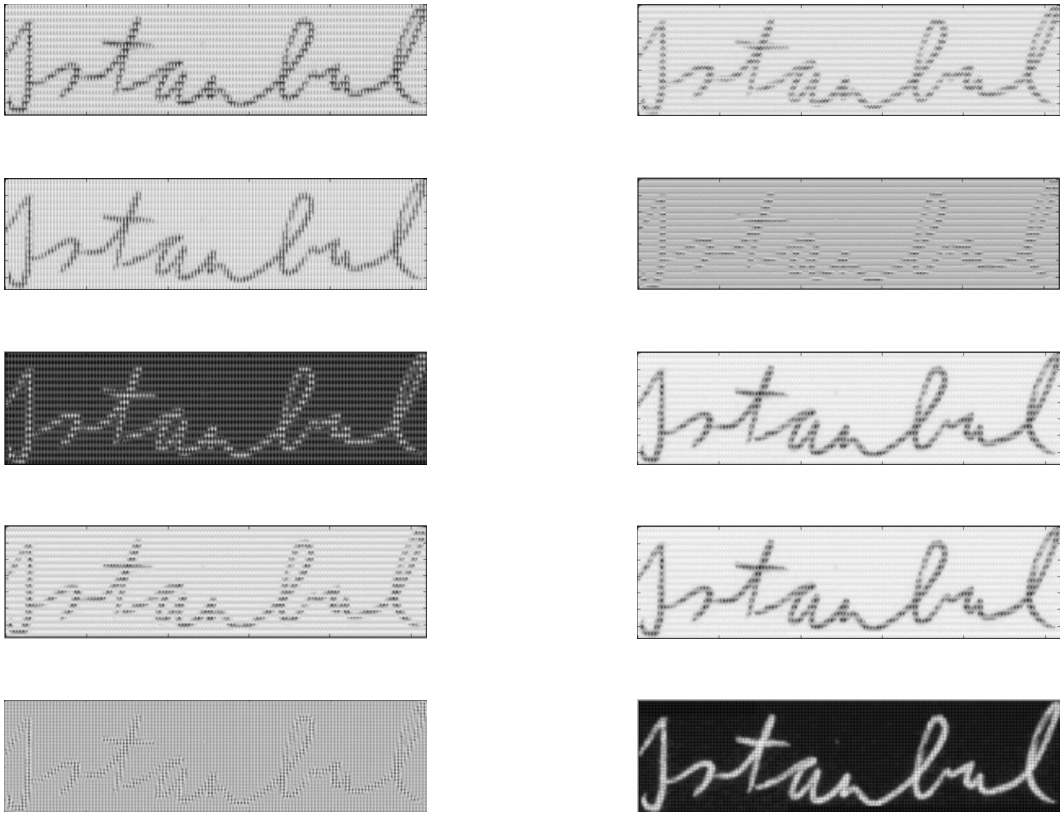

Figure 4.2: Filters of MDLSTM network.

Chapter 5

Conclusion and Future Work

The thesis investigated the application of Deep Neural Networks (DNN) for Handwriting Recognition in scanned documents and Automatic Speech Recognition (ASR) for people with speech impairment. We tried to tackle much more difficult problem than recognizing normal or noisy speech. We experimented with both traditional models like GMM-HMM and recent models like DNN-HMM, where in most of the cases DNN-HMM system was superior than GMM-HMM system. We tried two different types of adaptation techniques i.e speaker adaptation by transforming features using fMLLR transformation matrix and another is speaker adaptation by learning hidden unit contributions for CNN-RNN-CTC model. Since there is not much research in dysarthric speech recognition using deep neural networks, we show different methods for recognizing dysarthric speech given less amount of dysarthric speech data. In addition, we used DNNs i.e (MDLSTM) to recognize handwritten characters in scanned documents, which is far more complex than recognizing typed characters in scanned documents.

One of the major problems with recognizing dysarthric speech is inter speaker variability and the amount of data available for models to understand

patterns across speakers with speech impairment. Mengistu et al. [44] tackled these problems by using speaker adaptation technique like Maximum Likelihood Linear regression (MLLR) transformation followed by MAP adaptation using GMM-HMM, along with speaker specific dictionaries based on acoustic characteristics of each speaker. Since DNNs have performed better than GMMs on normal speech tasks, we experimented with DNNs for dysarthric speech. Our CNN-RNN-CTC model avoids the traditional feature extraction pipeline and directly uses spectrograms as an input. We found in order for the convolutional filters to learn more varied patterns across dysarthric speakers, we need more data than just dysarthric and control speaker's data from the TORGO database. Augmenting data using speed and tempo perturbation helps to reduce WER across dysarthric speakers by creating more variation, i.e. slow and fast speaking styles of dysarthric speakers. We further perform LHU adaptation which worked to reduce WER only for severely dysarthric speakers.

Another technique we used was the fMLLR transform, which is also known as constrained MLLR or feature space MLLR. Rather than adapting model parameters as we do in MAP adaptation and CNN-RNN-CTC, here we just estimate the transformation matrix for every speaker and normalize the features using this transformation matrix and use these features for training and testing. Based on the results in Table 4.2, this worked better on female speakers than on male speakers. One of the drawbacks with fMLLR is that it is an unsupervised adaptation method, where in order to estimate a trans-

formation matrix for a test speaker, an initial decoding is done on test data using a model that is not trained on fMLLR features. Once we get transcripts from decoding, they are used for estimating an initial fMLLR transofrmation matrix. One of the reasons why this works poorly is because initial decoding is done using a triphone model, which is not performing good. The triphone model is used to produce transcripts for an initial estimation of fMLLR transformation matrix.

From the results of the CNN-RNN-CTC and DNN Framewise models we can conclude that we cannot create a generalized model across dysarthric speakers. The dysarthric speakers have slower speaking rate than control speakers, some speakers tend to speak clearly till half part of the sentence and later half is usually unclear. Some have difficulty in speaking specific words mostly in long sentences, some speakers decrease their volume as breath support decreases while speaking. There are speakers with neck injury and speakers who cannot keep their neck or body stable while speaking. All these and other factors contribute to the variation of dysarthric speech. Generating and acquiring more data both for the CNN-RNN-CTC model in order to learn better CNN filters and the DNN Framewise model to better estimate fMLLR transformation matrix should help to improve recognition of dysarthric speech. Creating a speaker specific solution like custom dictionaries specific to each speaker based on their articulatory characteristics should be helpful.

In the future, we would like to use fMLLR transformation on features extracted from CNNs and use them for training a CNN-RNN-CTC model.

Also, modifying dysarthric speech waveform by correcting speech or mispronunciations or using visual information from lips or a mechanism to estimate articulatory filters that generate speech sounds from speech would be an interesting area of research. Another important area to investigate would be to get information from face of dysarthric speakers or through sensors around face. Dysarthric speech recognition is still an open area of research and we think in coming years we can see much more improvement in dysarthric ASR and human computer interaction for people with cerebral palsy or parkinson's or ALS.

For handwriting recognition, we developed an MDLSTM architecture for recognizing handwritten text, but our dataset consisted of segmented words, which were given as an input to the model. We showed that even though there are n number of different ways to write a same word, MDLSTM networks are able to recognize them. Future research could involve a model, where rather than just giving an image of a word to recognize handwritten text in it, we would give an entire sentence or an entire paragraph. Also, we would like to test the model on different languages like French, Hindi or Tamil and reduce computation time required to process images by implementing the entire architecture on a Graphic Processor Unit (GPU) instead of multiple cores, which is the current implementation.

Bibliography

[1] http://sox.sourceforge.net/main/homepage.

[2] https://swphonetics.com/praat/tutorials.

[3] Ossama Abdel-Hamid and Hui Jiang. Fast speaker adaptation of hybrid nn/hmm model for speech recognition based on discriminative learning of speaker code. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7942–7946. IEEE, 2013.

[4] Ossama Abdel-Hamid and Hui Jiang. Rapid and effective speaker adaptation of convolutional neural network based models for speech recognition. In *Interspeech*, pages 1248–1252, 2013.

[5] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. *arXiv preprint arXiv:1512.02595*, 2015.

[6] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[7] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for

measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013.

[8] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[9] Steven Davis and Paul Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE transactions on acoustics, speech, and signal processing*, 28(4):357–366, 1980.

[10] Li Deng, Ossama Abdel-Hamid, and Dong Yu. A deep convolutional neural network using heterogeneous pooling for trading acoustic invariance with phonetic confusion. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6669–6673. IEEE, 2013.

[11] P Enderby. Frenchay dysarthria assessment. *British Journal of Disorders of Communication*, 15(3):165–173, 1980.

[12] Salvador Espana-Boquera, Maria Jose Castro-Bleda, Jorge Gorbe-Moya, and Francisco Zamora-Martinez. Improving offline handwritten text recognition with hybrid hmm/ann models. *IEEE transactions on pattern analysis and machine intelligence*, 33(4):767–779, 2011.

[13] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[14] Sadaoki Furui. Cepstral analysis technique for automatic speaker verification. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29(2):254–272, 1981.

[15] Paul D Gader, Magdi Mohamed, and Jung-Hsien Chiang. Handwritten word recognition with character and inter-character neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 27(1):158–164, 1997.

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

[17] Mark JF Gales. Semi-tied covariance matrices for hidden markov models. *IEEE transactions on speech and audio processing*, 7(3):272–281, 1999.

[18] Roberto Gemello, Franco Mana, Stefano Scanzio, Pietro Laface, and Renato De Mori. Linear hidden transformations for adaptation of hybrid ann/hmm models. *Speech Communication*, 49(10):827–835, 2007.

[19] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

[20] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the*

23rd international conference on Machine learning, pages 369–376. ACM, 2006.

[21] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

[22] Hynek Hermansky. Perceptual linear predictive (plp) analysis of speech. *the Journal of the Acoustical Society of America*, 87(4):1738–1752, 1990.

[23] Geoffrey Hinton. A practical guide to training restricted boltzmann machines. *Momentum*, 9(1):926, 2010.

[24] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[25] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[26] Chen Huang and Sargur N Srihari. Word segmentation of off-line handwritten documents. In *Electronic Imaging 2008*, pages 68150E–68150E. International Society for Optics and Photonics, 2008.

[27] Zhen Huang, Sabato Marco Siniscalchi, I-Fan Chen, Jiadong Wu, and Chin-Hui Lee. Maximum a posteriori adaptation of network parameters in deep models. *arXiv preprint arXiv:1503.02108*, 2015.

[28] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[29] Johannes Jaschul. Speaker adaptation by a linear transformation with optimised parameters. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'82.*, volume 7, pages 1657–1660. IEEE, 1982.

[30] Alexander Kain, Xiaochuan Niu, John-Paul Hosom, Qi Miao, and Jan PH van Santen. Formant re-synthesis of dysarthric speech. In *Fifth ISCA Workshop on Speech Synthesis*, 2004.

[31] Alexander B Kain, John-Paul Hosom, Xiaochuan Niu, Jan PH van Santen, Melanie Fried-Oken, and Janice Staehely. Improving the intelligibility of dysarthric speech. *Speech communication*, 49(9):743–759, 2007.

[32] Naoyuki Kanda, Ryu Takeda, and Yasunari Obuchi. Elastic spectral distortion for low resource speech recognition with deep neural networks. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 309–314. IEEE, 2013.

[33] Ray D Kent, Gary Weismer, Jane F Kent, and John C Rosenbek. Toward phonetic intelligibility testing in dysarthria. *Journal of Speech and Hearing Disorders*, 54(4):482–499, 1989.

[34] Tom Ko, Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur. Audio augmentation for speech recognition. In *INTERSPEECH*, pages 3586–3589, 2015.

[35] Michał Kozielski, Patrick Doetsch, and Hermann Ney. Improvements in rwth's system for off-line handwriting recognition. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pages 935–939. IEEE, 2013.

[36] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[37] Honglak Lee, Peter Pham, Yan Largman, and Andrew Y Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in neural information processing systems*, pages 1096–1104, 2009.

[38] Li Lee and Richard C Rose. Speaker normalization using efficient frequency warping procedures. In *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, volume 1, pages 353–356. IEEE, 1996.

[39] Christopher J Leggetter and Philip C Woodland. Maximum likelihood linear regression for speaker adaptation of continuous density hidden markov models. *Computer Speech & Language*, 9(2):171–185, 1995.

[40] Xin Yu Liu and Michael Blumenstein. Experimental analysis of the modified direction feature for cursive character recognition. In *Frontiers in Handwriting Recognition, 2004. IWFHR-9 2004. Ninth International Workshop on*, pages 353–358. IEEE, 2004.

[41] Andrew L Maas, Ziang Xie, Dan Jurafsky, and Andrew Y Ng. Lexicon-free conversational speech recognition with neural networks. In *HLT-NAACL*, pages 345–354, 2015.

[42] U-V Marti and Horst Bunke. A full english sentence database for off-line handwriting recognition. In *Document Analysis and Recognition, 1999. ICDAR'99. Proceedings of the Fifth International Conference on*, pages 705–708. IEEE, 1999.

[43] Ofer Matan, Henry S. Baird, Jane Bromley, Christopher J. C. Burges, John S. Denker, Lawrence D. Jackel, Yann Le Cun, Edwin P. D. Pednault, William D Satterfield, Charles E. Stenard, et al. Reading handwritten digits: A zip code recognition system. *Computer*, 25(7):59–63, 1992.

[44] Kinfe Tadesse Mengistu and Frank Rudzicz. Adapting acoustic and lexical models to dysarthric speech. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 4924–4927. IEEE, 2011.

[45] Yu Nakajima, Shunji Mori, Shuuki Takegami, and S Sato. Global methods for stroke segmentation. *International Journal on Document Analysis and Recognition*, 2(1):19–23, 1999.

[46] Toru Nakashika, Toshiya Yoshioka, Tetsuya Takiguchi, Yasuo Ariki, Stefan Duffner, and Christophe Garcia. Dysarthric speech recognition using a convolutive bottleneck network. In *Signal Processing (ICSP), 2014 12th International Conference on*, pages 505–509. IEEE, 2014.

[47] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 5206–5210. IEEE, 2015.

[48] Erwan Pépiot. Male and female speech: a study of mean f0, f0 range, phonation type and speech rate in parisian french and american english speakers. In *Speech Prosody 7*, pages 305–309, 2014.

[49] Erwan Pépiot. Voice, speech and gender:. male-female acoustic differences and cross-language variation in english and french speakers. *Corela. Cognition, représentation, langage*, (HS-16), 2015.

[50] Vu Pham, Théodore Bluche, Christopher Kermorvant, and Jérôme Louradour. Dropout improves recurrent neural networks for handwriting recognition. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 285–290. IEEE, 2014.

[51] Réjean Plamondon and Sargur N Srihari. Online and off-line handwriting recognition: a comprehensive survey. *IEEE Transactions on pattern analysis and machine intelligence*, 22(1):63–84, 2000.

[52] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. The kaldi speech recognition toolkit. In *IEEE 2011 workshop on automatic speech recognition and understanding*, number EPFL-CONF-192584. IEEE Signal Processing Society, 2011.

[53] Shakti P Rath, Daniel Povey, Karel Veselỳ, and Jan Cernockỳ. Improved feature processing for deep neural networks. In *Interspeech*, pages 109–113, 2013.

[54] Douglas A Reynolds, Thomas F Quatieri, and Robert B Dunn. Speaker verification using adapted gaussian mixture models. *Digital signal processing*, 10(1):19–41, 2000.

[55] Frank Rudzicz. Acoustic transformations to improve the intelligibility of dysarthric speech. In *Proceedings of the Second Workshop on Speech and Language Processing for Assistive Technologies*, pages 11–21. Association for Computational Linguistics, 2011.

[56] Frank Rudzicz, Aravind Kumar Namasivayam, and Talya Wolff. The torgo database of acoustic and articulatory speech from speakers with dysarthria. *Language Resources and Evaluation*, 46(4):523–541, 2012.

[57] Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural

network training with high-dimensional output targets. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6655–6659. IEEE, 2013.

[58] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[59] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.

[60] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[61] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[62] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using lstms. In *ICML*, pages 843–852, 2015.

[63] Gilbert Strang. Introduction to linear algebra. 2011.

[64] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[65] Pawel Swietojanski and Steve Renals. Learning hidden unit contributions for unsupervised speaker adaptation of neural network acoustic models. In *Spoken Language Technology Workshop (SLT), 2014 IEEE*, pages 171–176. IEEE, 2014.

[66] Øivind Due Trier, Anil K Jain, and Torfinn Taxt. Feature extraction methods for character recognition-a survey. *Pattern recognition*, 29(4):641–662, 1996.

[67] Karel Veselỳ, Arnab Ghoshal, Lukás Burget, and Daniel Povey. Sequence-discriminative training of deep neural networks. In *Interspeech*, pages 2345–2349, 2013.

[68] SG Webber. Webber photo cards: Story starters. 2005.

[69] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[70] Phil C Woodland. Speaker adaptation for continuous density hmms: A review. In *ISCA Tutorial and Research Workshop (ITRW) on Adaptation Methods for Speech Recognition*, 2001.

[71] Alan Wrench. The mocha-timit articulatory database, 1999.

[72] Jian Xue, Jinyu Li, and Yifan Gong. Restructuring of deep neural network acoustic models with singular value decomposition. In *INTERSPEECH*, pages 2365–2369, 2013.

[73] Jian Xue, Jinyu Li, Dong Yu, Mike Seltzer, and Yifan Gong. Singular value decomposition based low-footprint speaker adaptation and personalization for deep neural network. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6359–6363. IEEE, 2014.

[74] Kathryn M Yorkston and David R Beukelman. *Assessment of intelligibility of dysarthric speech*. Pro-ed, 1984.

[75] Dong Yu, Kaisheng Yao, Hang Su, Gang Li, and Frank Seide. Kl-divergence regularized deep neural network adaptation for improved large vocabulary speech recognition. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7893–7897. IEEE, 2013.

[76] Victor Zue, Stephanie Seneff, and James Glass. Speech database development at mit: Timit and beyond. *Speech Communication*, 9(4):351–356, 1990.