Real-time Acoustic Modeling with Convolutional Neural Networks

Ying Hang Seah, Suvir Mirchandani, Levi Lian
{yinghang, smirchan, levilian}@stanford.edu

Abstract

Acoustic modeling with Hidden Markov Models and Gaussian Mixture Models has been the standard approach for automatic speech recognition (ASR) until the introduction of Convolutional Neural Networks (CNNs). We investigate the use of CNNs for a smaller task—phoneme recognition—and extend the model to allow for real-time classification. The real-time nature of the task poses challenges for streaming both the input and the output. We show that the CNN is able to produce decent performance for audio inputs given its unique characteristics. Additionally, we adapt the real-time classification task to streaming data visualization. This provides a base for a phoneme practicing tool that can be used by people with speaking difficulties. Future research can improve the usability of this system and extend the approach beyond English phonemes. We have open-sourced the code for this project.[1].

1 Introduction

Acoustic modeling is a technique for determining the phonemes in audio input of speech [1]. It is commonly used in conventional automatic speech recognition (ASR) systems to generate a series of phonemes, which is then fed into a pronunciation model, and then a language model to create a transcription of the audio input [2]. Instead of performing speech recognition, this project's goal is to create an acoustic model for a streaming algorithm, thus enabling phoneme recognition to be performed in real-time.

The motivation for our project arises from the need to create a practicing tool for people who are facing difficulties in speaking English. This includes individuals who are learning English as a second language and dysarthric patients with motor-cognitive speech impairments [3]. Dysarthria, which results from stroke and other brain damage, weakens the vocal muscles and can affect the intelligibility of a person's speech. Treatment for dysarthria includes practicing overarticulation of phonemes with a speech-language pathologist. By performing real-time phoneme recognition, our proposed algorithm could help dysarthric patients practice their articulation and get immediate feedback without the need for a human listener.

Given the recent success of Convolutional Neural Networks (CNN) in the field of computer vision, we investigate the performance of phoneme recognition using feed-forward CNN without the use of attention.

2 Related Work

Most acoustic models use a combination of Gaussian Mixture Models (GMM) with Hidden Markov Models (HMM) to model the sequence of phonemes from the audio input [2], [4]. The GMM-HMM approach went into popularity around the same time when Mel Frequency Cepstral Coefficients (MFCCs) became the standard features for audio

[1]Our code repository is located at `bit.ly/221-phoneme-repo`. An interactive worksheet for reproducibility purposes is available at `bit.ly/221-phoneme-wkst`.

CS221: Artificial Intelligence, Fall 2018, Stanford University, CA.

inputs. MFCCs are a set of features that use decorrelated filterbank coefficients. Log Filterbank Energies (LFBs), from which these coefficients are extracted, have been the standard features used in speech processing, dated as early as to 1975 [5]. The motivation for filterbanks as features is intuitive: they are designed to capture the nature of the nonlinear human perception of sound. Filterbanks first apply the Hamming window function to audio frames, and then utilize the Short-Term Fourier Transform and power spectrum. Finally, triangular filters are employed to mimic the discriminative hearing range of humans.

MFCCs, however, conduct an additional step of processing. They decorrelate the filterbank coefficients and yield a compressed representation of them. If the coefficients are correlated, the correlations among features would hinder effective implementation of machine learning models. The introduction of MFCCs can thus be seen as a necessary compromise for the inherent problems with machine learning [6].

It is only in recent years that CNNs become a popular model for the audio input after the stunning success with the ImageNet challenge [7]. Recent research has shown that deep neural networks actually outperform GMMs on a variety of speech recognition benchmarks [8]. CNN was introduced and successfully applied on the TIMIT database by [9] using layers of CNN and Connectionist Temporal Classification (CTC) where they achieved a phone error rate (PER) of 18.2%, similar to the PER in BiLSTM sequence models.

The aforementioned limitations with MFCCs do present challenges for new methods, such as CNNs. Because neural networks are less susceptible to highly correlated input, the additional transformations in MFCCs appear unnecessary. They may even generate over-simplification, as the discrete cosine transform (DCT) applied on MFCCs is a linear transformation, which discards hidden relationships encoded in nonlinear speech signals.

Therefore, when it comes to CNNs, researchers have resorted to LFBs and their variants, though MFCCs are still valid features worthy of experimentation. Using LFBs or MFCCs and their first and second derivatives as feature layers has also been proposed as an even more expansive feature set that captures the rate of change over time [6]. No matter which feature set is used, the general idea with CNNs applied the audio input is standard. Convolution is applied on the frequency axis of the spectrogram, modeling the spectrogram as an image. This minimizes the impact of unwanted small shifts in frequency on the learned features. This practice improves the CNN structure for TIMIT phone recognition, with a relative PER reduction of about 8.5%.

Additionally, with the introduction of new techniques in computer vision such as Batch Normalization [10] to normalize inputs between layers, we incorporate these new components into our models. We analyze how different architectures affect the PER, and to meet our end goal, we develop a real-time practicing tool with graphical output.

3 Problem Definition

We formulate our problem as follows. Given an audio stream S, a set of phonemes P, and a training set $\mathcal{D}_{train}$ which contains audio files labeled with phonemes, train a function f to classify the phoneme $p \in P$ being uttered at frame t in S. Additionally, we wish to visualize the output from f in real-time.

At a high level, we are creating a system that takes in a streaming audio input and outputs the vowel phoneme being uttered, with real-time updates.

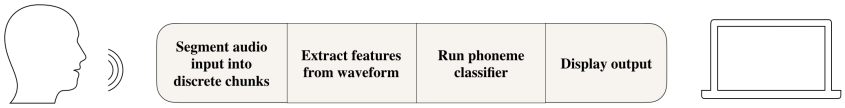

Figure 1: An overview diagram of the system.

In terms of visualization, the main objective, as stated above, is to visualize the output in real-time. This entails two things from the end user's point of view. First, the output should be made understandable for the user without causing too much distress. Thus, ti enhance readability of graphs, comparison with the ground truth, additional labeling and feature representations, and clear legends are needed. Notably, converting an audio file to an image format would help the most. Second, the whole graphical generative process needs to take place in a responsive manner. Drawing out the whole user experience pipeline and conducting user research would help initiate the experiential starting points and iterate based on user feedback.

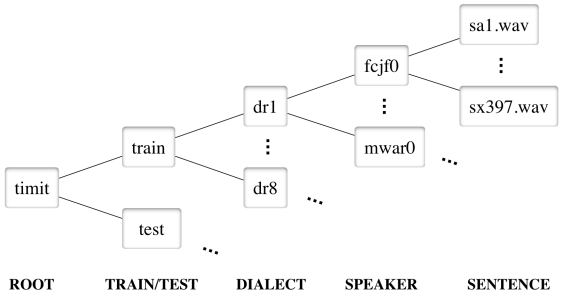

Figure 2: An illustration of the directory structure of TIMIT.

4 Approach

4.1 Infrastructure

4.1.1 Dataset

We use the TIMIT corpus, a dataset of sentence recordings spoken by speakers of different dialects of American English. A common dataset for speech recognition models, TIMIT is particularly useful for our purposes because it includes hand-verified, time-aligned phonetic annotations. We acquired access to the TIMIT corpus via Stanford's Linguistics Department.

TIMIT is organized as illustrated in Figure 2. At the top level, it is split into train and test directories. In each directory, there are eight English dialects, and within each dialect there are directories for speakers. For each speaker, there are ten audio files of sentences that they recorded.

We wrote a data parsing program that a) preprocesses the audio using `sox`[2], b) slices the audio files into phonemes, and c) computes features from each audio file. 10 contiguous frames are selected from sliced phoneme files that are longer than 10 frames; those that are shorter than 10 frames are expanded about the center to this length. Section 4.2 discusses feature extraction in further detail.

4.1.2 Real-time Application

Although our training and testing data consists of discrete audio files for each phoneme, our end goal is to use this classifier in the context of a real-time feedback application. To this end, part of our infrastructure includes reading a real-time audio stream (using the Python module `PyAudio`[3]) and then parsing it into discrete intervals that can be passed into our classifier, evaluated, and have the results displayed in real-time. We discuss this process in more detail in Section 5.3.

4.2 Feature Extraction

We identify numerical measurements that can be extracted as features from the waveform which represent the way that humans interpret sound. These features could thus allow a classifier to distinguish sounds loosely to the same ability as a human.

One of the features we investigate is Mel Frequency Cepstral Coefficients (MFCCs). As mentioned in Section 2, these coefficients are a common choice for features in speech recognition. MFCCs are calculated by applying a filterbank to an estimate of the power spectrum (calculated over certain windows), and taking the logarithm followed by discrete cosine transform of the filterbank energies [11]. MFCCs loosely represent the sounds of speech as manifested in human ears and account for our lack of ability to distinguish linear differences in the frequency. We use a frame size of 20ms and a stride size of 5ms.

Another feature we look at is the log filterbank energies (LFB) of the waveform, which are calculated for MFCCs prior to the discrete cosine transform. Additionally, we try augmenting log filterbank energies with delta and delta-delta values (LFB+D). The intuition behind adding the channels' delta and delta-delta is that our ears are tuned to hear

[2] http://sox.sourceforge.net

[3] https://people.csail.mit.edu/hubert/pyaudio/

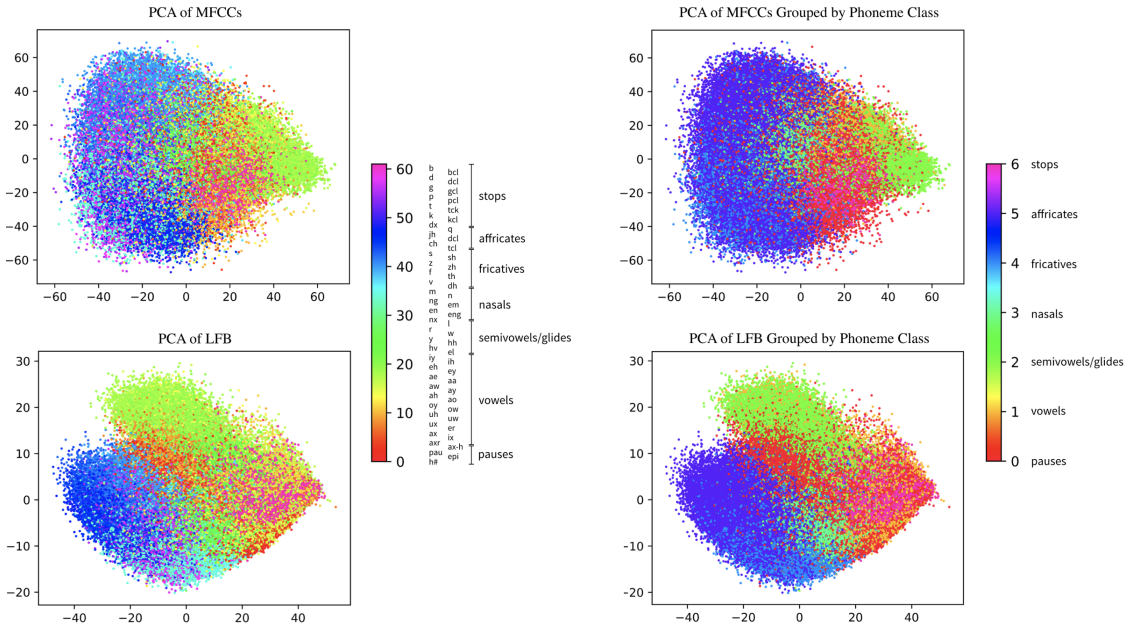

Figure 3: Visualization of TIMIT in the MFCC and LFB feature space after Principal Component Analysis. Two plots of each are shown; one with each phoneme assigned to a color, and one with each phoneme class assigned to a color.

changes rather than merely static frequency, and using gradients as a feature will enable the model to extract changes in frequency.

As a way to visualize the data, we perform Principal Component Analysis on the MFCC and LFB features for dimensionality reduction. The results of the PCA are shown in Figure 3 (i.e., all the phoneme files derived from TIMIT plotted in the reduced feature space). On the left, the phonemes are colored based on their order in the TIMIT documentation. We observed that the different classes of phonemes (stops, affricates, etc.) are located in different locations of the feature space for both MFCCs and LFB. Thus, on the right, we discretize the colors into groups based on the phoneme classes.

This visualization is informative because it indicates that MFCCs and LFBs are generally good features to separate phonemes; phonemes from different classes should indeed be separated in the feature space. Phonemes in the same class (e.g., `m` and `n`, which are both nasals) are close to each other in the feature space, as expected.

Figure 4 shows examples of MFFCs and LFB values plotted for four instances of a particular phoneme in TIMIT.

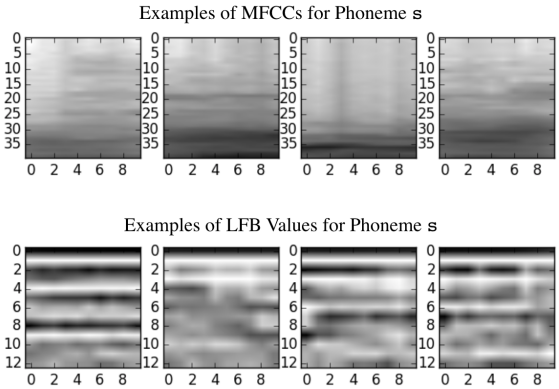

Figure 4: Examples of MFCCs and LFB values for four occurrences of `s` in TIMIT. The horizontal axis is time and the vertical axis are the feature values at a particular frame.

h# pau	ih ix	s	sh	axr er r	l	z	f	eng ng	uh ux	aa ao	ow	iy	ae	eh	n	m

Table 1: The different phoneme groupings that we use as classes.

4.3 Constraining the Problem

Rather than including all 63 phonemes in the TIMIT dataset, we constrain the problem to 17 phoneme groups. We devised this constrained problem by ordering the phonemes by their frequency in TIMIT and then analyzing the continuous phonemes (which are more amenable to a live feedback system). Phonemes that we deemed to sound the same (or extremely similar) were grouped together. We disregarded closures, the schwa, semivowels, glides, epenthetic silence, affricates, and nasal flaps. The result is shown in Table 1.

4.4 Classification Models

4.4.1 Evaluation Metric

To evaluate performance on the classification task, we are using a conventional metric in acoustic modeling: the phoneme error rate (PER). It is calculated as follows:

$$\text{PER} = \frac{W}{T} \tag{1}$$

where W = total number of wrong predictions and T = total number of predictions made.

4.4.2 Baseline

To gain an intuition for the lower bound on the performance of the problem, we used a simple multi-class classification (logistic regression) model that uses logartihmic softmax as the output. We are using cross-entropy loss. It uses the LFB corresponding to the middle of the audio clip as a feature vector. The phoneme error rate of the baseline is 0.3160.

4.4.3 Oracle

To understand the upper bound on the performance, we conducted a small human test to serve as an oracle. an English native speaker student with good hearing performed the oracle. He randomly selected a clip from each class as "training" examples, and then hand-annotated 50 random vowel clips from the dataset into each class. An annotation was counted as correct if the human annotation matched the true TIMIT label. The phoneme error rate of the oracle is 0.12. The difference in performance between the oracle and baseline indicates that there is potential for a more sophisticated solution that can bridge the gap in performance.

4.4.4 Convolutional Neural Networks

The model we focus on is a 2D CNN that combines convolutional layers, a max-pooling layer, non-linear activation functions, fully-connected layers and a logarithmic softmax based on [9]. A cross-entropy loss function is used as we are performing multi-class classification.

The input of the model is a 2D grid where the vertical axis is the frequency band and the horizontal axis is the frame number, given by MFCC and LFB. For the output, we have 17 probabilities with each corresponding to the different phoneme classes, including a "silent" phoneme which is used when there are no phonemes present.

As shown in Figure 5, we experimented with three different CNN architectures with various depths of convolutional layers: 2 layers of convolution, 5 layers of convolution, and 10 layers of convolution. The structure of the 10-Layer CNN is adopted from [9], while the 5-Layer CNN and the 2-Layer CNN are simplified models with fewer layers. Experimenting with models of varying degrees of complexity allows us to find out the minimum complexity required for a model to perform equally well as the model from [9]. Simpler models mean less computation, enabling faster inference to provide real-time feedback.

In all three architectures, the convolutional layers have a filter size of 5×3. We use a max-pooling layer with size of 1×3 only after the first convolutional layer. This means the max-pool is only applied along the time scale. Batch normalization is also applied every two layers to reduce internal covariate shifts within the model.

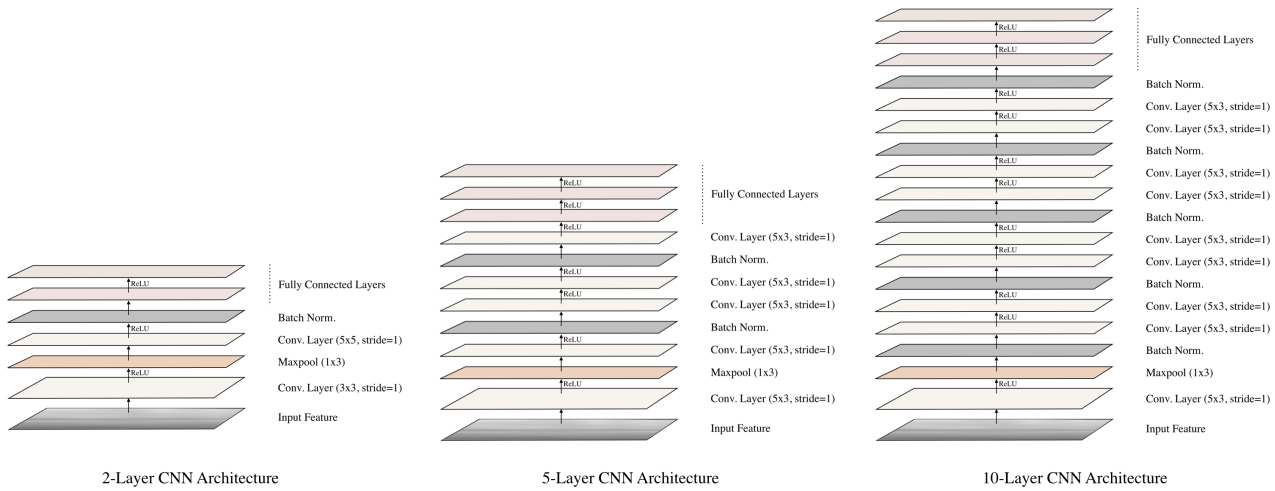

Figure 5: Illustration of the various layers in our three CNN architectures.

We describe these different CNN components in more detail below.

Convolution Based on [9], we explain the function of the first convolutional layer. Let $\mathbf{X} \in \mathbb{R}^{b \times t}$ be a spectrogram feature with frequency bandwidth b (e.g., the number of elements in one frame of the MFCCs), and number of frames t. This layer convolves $\mathbf{X}$ with k filters $\mathbf{W_i}_k$ where each $\mathbf{W_i} \in \mathbb{R}^{m \times n}$ has height m (along the frequency axis) and width n (along the frame axis).

Max-pooling The max-pooling layer is used to output the maximum unit from j adjacent units and therefore make the input dimension smaller. We perform pooling along the time axis. This max-pooling is applied only once after the first convolutional layer, not every single convolutional later because as explained by [9], "as more pooling layers are applied, units in higher layers would be less discriminative with respect to the variations in input features."

Batch Normalization Batch normalization first normalizes each dimension of a mini-batch of outputs from the previous layer to have zero mean and unit variance. After normalization, each dimension d is scaled by γ_d and shifted by β_d. γ and β are parameters that the network can learn to optimize.

Rectified Linear Unit (ReLU) ReLU is an activation function. It is mathematically defined as $f(x) = \max(0, x)$. It avoids and rectifies the vanishing gradient problem and is less computationally expensive than other functions such as sigmoid.

Cross-Entropy Loss Cross-entropy loss measures the performance of a classification model where the output is a probability between 0 to 1. The loss increases as the prediction of the output diverges from the label. Since we are performing multi-class classification, where $C = 17$, we calculate a separate loss for each label and sum all the losses. Cross-entropy loss is calculated by

$$H(p, q) = -\sum_{c=1}^{17} p_c \log(q_c) \tag{2}$$

where p_c is the actual label index and and q_c is the probability estimate generated by the model.

Adam Optimizer For each mini-batch of data, which in our case is 20 examples, we used Adam Optimizer [12] to update the network weights using first-order and second-order momentum of the gradients with decay to enable faster convergence for stochastic gradient descent.

Randomized Hyperparameter Search We implemented a hyperparameter sweep for the learning rate, L2 regularizer and dropout rate. Both the learning rate and L2 regularizer are randomly generated on an exponential scale while the dropout rate is randomly generated on a linear scale. Sample results for a hyperparameter sweep are shown in Figure 6.

Classifier	PER
Logistic Regression (Baseline)	0.3160
Oracle (Human Annotation)	0.12
2-Layer CNN	0.2008
5-Layer CNN	0.2042
10-Layer CNN	0.2038

Table 2: Best PER for Different Models Architectures using LFB Features.

Feature	PER
MFCC	0.2780
LFB	0.2008
LFB+D	0.1953

Table 3: Best PER for 2-Layer CNN Architecture using Different Features.

5 Results and Discussion

5.1 Classifier Performance

We evaluate the classifiers using a hierarchical approach. We first test the different architectures, comparing them to our baseline and oracle, and then investigate different features and their effect on classifier performance.

Table 2 displays the results of the different classifiers using LFB features. All of the CNN architectures outperformed our baseline (logistic regression), reducing the error rate by at least one third. The 2-Layer CNN marginally outperformed the 5-Layer and 10-Layer architectures, telling us that the simplest of the CNN models was indeed sufficient to achieve about 80% accuracy.

The results in Table 2 were the outcome of the hyperparameter search described in Section 5.2.1. We show an example of the hyperparameter search in Figure 6 and discuss this process further in Section 5.2.

With these results comparing the CNN architectures, we perform additional experiments with the 2-Layer CNN by altering the feature used (Table 3). We see that LFB outperforms MFCCs, and that LFB+D marginally outperforms LFB. We analyze these results in Section 5.2.

5.2 Analysis

5.2.1 Hyperparameter

We conducted a relatively exhaustive search on multiple combinations of hyperparameters. The results are shown in Figure 6. We find that learning rate correlates the most with the PER, which is justified in prior literature [13]. The learning rate shows us that the model initially gets stuck in local minima as the step size is too small. As we increase the learning rate, we gradually get to the global minima. However, as we further increase the rate, the model tends to overshoot, giving us a worse performance.

5.2.2 Initial Insights

Table 2 tells us that all three types of CNN outperform the baseline, improving performance by about a 36.7% relative error reduction over the baseline. The 2-Layer CNN performed slightly better than the multi-layered CNNs. At an abstract level, this might be explained by Occam's razor, which tells us that unnecessarily complex models generally perform worse.

Table 3 shows how different features result in different error rates. In particular, LFB+D performs the best, with MFCC giving the worst error rate.

Based on this result, we have gleaned two insights, which are presented below to support our decision of using CNNs.

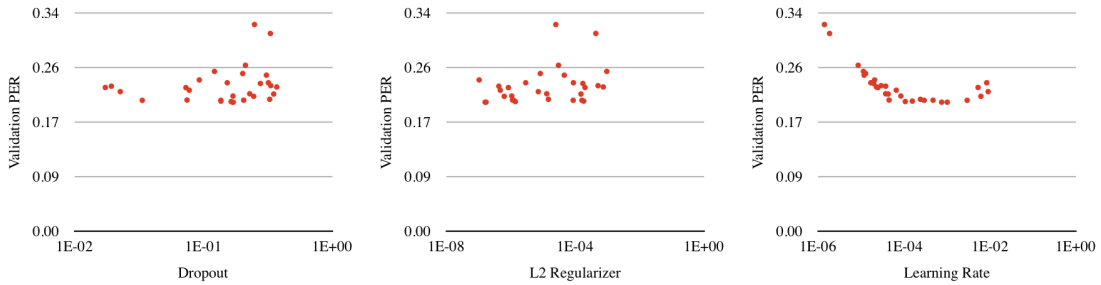

Figure 6: Validation PER for 2-Layer CNN with LFB features across 30 different hyperparameter configurations. Note that the hyperparameter scale is logarithmic.

Localities Speech spectrograms contain locality characteristics along the frequency axis, which the CNN picks up through max-pooling and limited weight sharing. Because different phonemes have energy concentrations in different local bands along the frequency axis, the local concentrations are used to help distinguish different phonemes, therefore generating better performance.

This characteristic of speech signals, however, entails that speech inputs come in a frequency scale that can be segmented into regions. Therefore, the MFCC features, which are merely docorrelated coefficients, are not suitable for the task; the DCT-based decorrelation transform would remove the local dependencies. On the other hand, filterbank features can work for the CNN configuration. This explains why we see better performance with LFB than MFCCs. LFB+D performs slightly better because it takes into account additional first temporal derivatives.

Robustness to Noise CNN also achieves speaker invariance and noise robustness. Because the diverse nature of the speakers, their varied frequencies theoretically would present difficulties for the model to learn. Part of the problem is solved once we introduce the local filters (i.e., max-pooling and limited weight sharing). Since the filters consider the local regions, they can both minimize the impact of ambient noise acting as outliers and maximize the influence of clean parts of the spectrum to detecting features. Also, the weight sharing and pooling mechanisms in CNN greatly reduce the number of parameters, decreasing training time.

5.2.3 Error Analysis

The confusion matrix in Figure 7 summarizes the prediction accuracy rate for our 17 classes for the best model (2-Layer CNN with LFB+D features).

Many of the errors are understandable. For instance, most of the phonemes with the accuracy rate lower than 60% do sound phonetically similar to other phonemes present. The predictions do support this trend. For example, the pairs (`n, eng`), (`s, z`), and (`ih, eh`) all have very similar sounds. Most of the errors shown are not necessarily a bad thing, because even humans would probably mistake those classifications. That being said, a user-centered practicing tool should be able to decipher small differences in phonemes to truly aid in pronuncation. It could also put the phonemes in word contexts to make them more distinguishable.

Notably, the training dataset is skewed toward certain phonemes, as shown in Figure 8, with `ux, uh` and `eng` having only one third of the largest phoneme data. This further clouds the correct prediction of already similar phonemes.

5.3 Visualization

We create a real-time visualization for the feature extractor and classifiers (Figure 9). This application lays the groundwork for a feedback system to assist with pronunciation. Short intervals are extracted from an audio stream, run through our feature extractor, and then evaluated using the learned classifier model. The results are then displayed on a spectrogram-like map for the features and a heat map for the classifier. We show the confidence of each class's prediction as an intensity.

6 Conclusion

The primary objectives of this paper are to demonstrate and justify the usefulness of CNN for acoustic modeling, specifically for the challenge of classifying phonemes in real-time, and to present data visualization for the purpose of

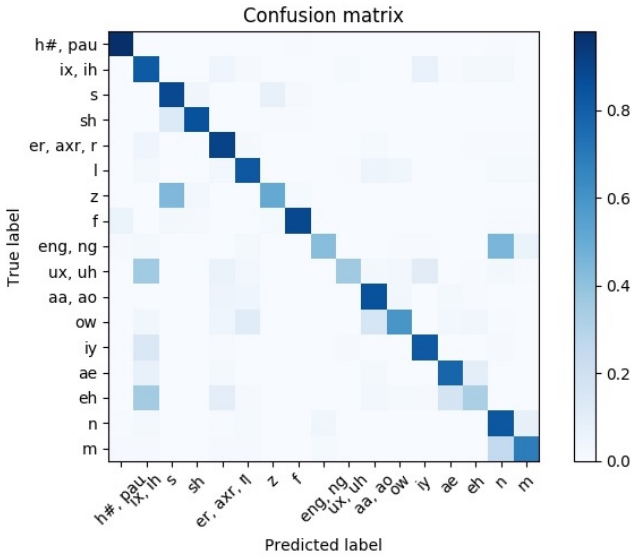

Figure 7: Confusion matrix for 2-Layer CNN with LFB+D features.

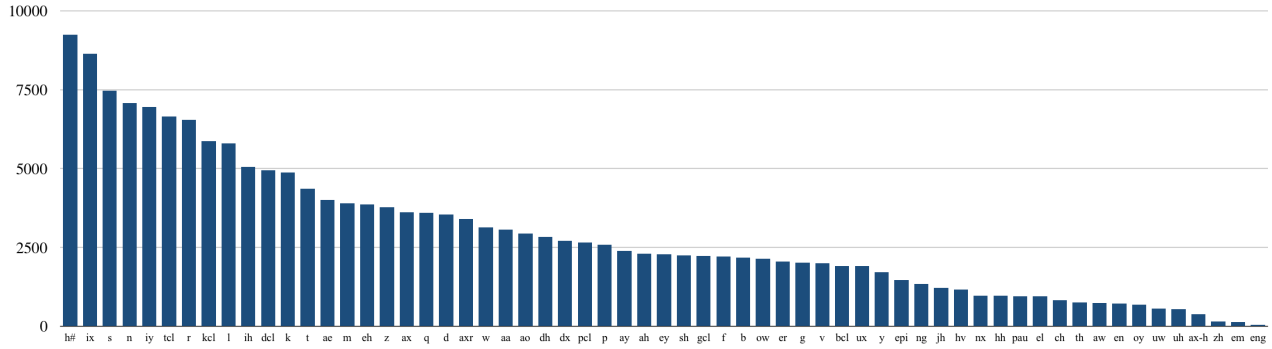

Figure 8: Frequency of phonemes in TIMIT.

practicing pronunciations. We first show how to model the CNN for the audio inputs from the TIMIT dataset. Second, we evaluate the performance of different model architectures given different hyperparameters and features. We show that a simple 2-Layer CNN is able to achieve the best result: 19.53% PER rate with LFB+D. We ascribe the success mainly to the localities of speech frequencies and the robustness to noise thanks to the unique characteristics of CNNs and LFBs. Finally, we showcase intuitive, user-centric real-time data visualizations to demonstrate the intended user experience.

Given the results, analysis, and relevant discussions, future work can be divided into two categories. The first is on better interpretability of models. Interpretability could provide the end-user with the necessary explanations for why their pronunciation is wrong, and offer suggestions to capture the key features of the pronunciation. Future research therefore should investigate quantitatively the influence of localities and robustness to noise on the model performance by setting up A/B experiments. By understanding the effects quantitatively, model performance and data visualization can be further improved.

Second, the TIMIT dataset should be better understood and the generalizability of our approach to other corpora is worth studying. TIMIT contains a wide variation of dialects that could potentially affect the frequency invariance assumption of the phonemes. So, using other corpora that is for sure unaffected by geographic differences is a possible direction. More importantly, we need to confirm if our findings are generally applicable to other languages with different features. In this way, more users would benefit from a general phoneme practicing tool.

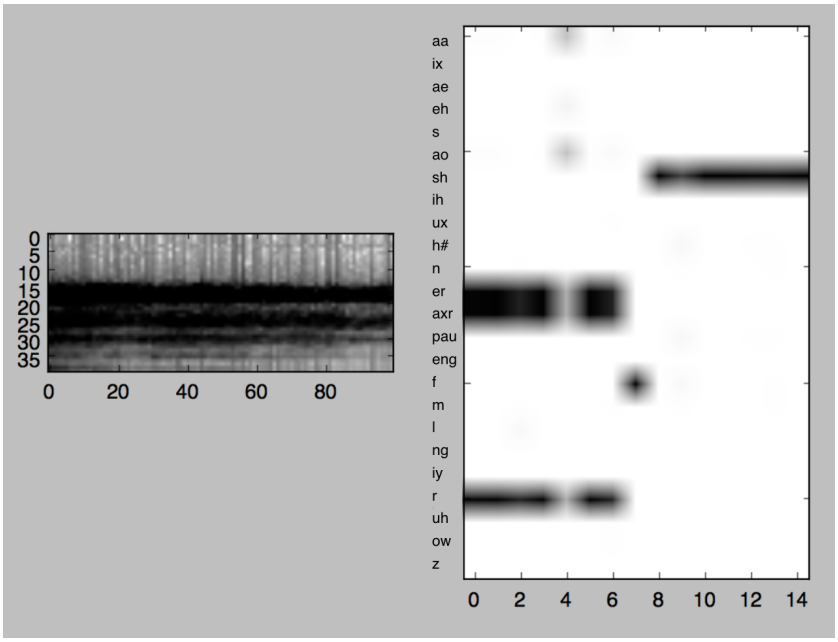

Figure 9: An sample of our real-time visualization. LFB feature extraction is displayed on the left and phoneme classification confidences are represented on the right.

References

[1] P. Coxhead, *Natural language processing & applications-phones and phonemes*, 2006.

[2] B. Gold, N. Morgan, and D. Ellis, *Speech and audio signal processing: processing and perception of speech and music*. John Wiley & Sons, 2011.

[3] J. R. Duffy, *Motor Speech Disorders-E-Book: Substrates, Differential Diagnosis, and Management*. Elsevier Health Sciences, 2013.

[4] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, *et al.*, "The kaldi speech recognition toolkit," in *IEEE 2011 workshop on automatic speech recognition and understanding*, IEEE Signal Processing Society, 2011.

[5] V. Lesser, R. Fennell, L. Erman, and D. Reddy, "Organization of the hearsay ii speech understanding system," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 23, no. 1, pp. 11–24, 1975.

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

[7] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. DOI: 10.1007/s11263-015-0816-y.

[8] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[9] Y. Zhang, M. Pezeshki, P. Brakel, S. Zhang, C. L. Y. Bengio, and A. Courville, "Towards end-to-end speech recognition with deep convolutional neural networks," *arXiv preprint arXiv:1701.02720*, 2017.

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

[11] M. Xu, L.-Y. Duan, J. Cai, L.-T. Chia, C. Xu, and Q. Tian, "Hmm-based audio keyword generation," in *Pacific-Rim Conference on Multimedia*, Springer, 2004, pp. 566–574.

[12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[13] D. R. Wilson and T. R. Martinez, "The need for small learning rates on large problems," in *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, IEEE, vol. 1, 2001, pp. 115–119.