Phoneme Classification with Multi-Headed Convolutional Neural Network

Merlin Carson
Department of Computer Science
Portland State University, Portland, OR, USA
Email: mpc6@pdx.edu
code repository: github.com/mpc6/PhonemeID

Abstract—**Phoneme classification is a common task in automatic speech recognition because it reduces the dimensionality of the prediction space from thousands of words in a dictionary to a few dozen sounds that are the building blocks for those words. This significantly reduces the required complexity of a speech recognition model. Current state of the art phoneme classification is achieved using bi-directional recurrent neural networks since the signal length of a spoken phoneme varies, and knowing how the signal is changing both forward and backward allows models to have insight towards what the current predicted phoneme should be by taking into consideration both past and future information. While recurrent neural networks perform well on time series data, due to their ability to track state, their internal structure is that of multiple fully connected layers, thus the spatial locality of the signal is lost through its propagation to all neurons in the layer. Modern speech processing algorithms like WaveNet adopt convolutional neural networks for tasks on audio-based, time series data, thus keeping some spatial locality to the processing of the signal through the network due to the shared parameters and sliding filter properties of the convolution operation. Therefore, in this paper convolutional neural networks for the task of phoneme classification are used so the model can learn and predict from the structure that exists in spectrograms created through transformations on raw audio data.**

Index Terms—**convolutional neural network, phoneme classification, Timit dataset**

I. Introduction

Using convolutional neural networks (CNNs) for audio-based, time series data has been a hot topic since Google released its WaveNet research [1] in 2016. The use of causal convolution layers with dilated filters have shown to significantly increase the receptive field of convolutional layers, allowing the model to make predictions based on information from a significant number of past time steps. Traditionally this type of temporal data modeling was done with recurrent neural networks, however, the thousands of sequences of timesteps required to model just a few hundred milliseconds of audio in the time domain created significant problems when backpropagating recurrent units through time. The inner workings of recurrent layers also consist of several parallel fully connected layers, which are computationally expensive and exponentially explode the parameter space. Convolutional layers work through parameter sharing, significantly reducing the number of trainable parameters, and thus are more size and time efficient.

For phoneme classification, long, time-based, stateful sequences are not required. Additionally, through a few pre-processing steps, the audio segment containing the phoneme utterance can be converted to the frequency domain to obtain Mel-Frequency Cepstral Coefficients (MFCCs) [2] which results in a significant reduction in dimensionality of data compared to the original time-domain audio. This allows for the features of an entire uttered phoneme to be processed in parallel, removing the need for a state based model and the computational complexity of bi-directional RNN layers that are used in current state of the art phoneme classifiers, which require the entire signal to be passed both forward and backward through the layer. In practice, a bi-directional RNN can start classifying a signal as it's arriving, but the confidence in its prediction will not be maximized until the entire signal has passed through the layer. Due to this phenomenon, and a significantly smaller number of parameters, a convolutional neural network should only require a single forward pass of the data to have all the information to confidently predict a phoneme from an input vector. Thus, resulting in less latency and more confidence in each prediction.

Since transforming time-domain audio to the frequency domain results in 2D data, magnitude of frequency bins by a number of discrete time intervals, it can be viewed like an image as a spectrogram. Therefore, latent features can be learned from the pre-processed data using the local area spatial locality and translational invariant properties of convolutional neural network layers. This allows the model to interpret how the input signal is changing over time, without it requiring the changes to occur at the same time or intensity in the audio input vector.

For most speech-centric processing applications, raw audio is transformed into MFCCs to best capture the characteristics of speech. In addition to this feature, the first and second deltas of the MFCCs can inform a model of the rate of change of the MFCCs, and thus how the speech is changing over time. And finally, distance between MFCC vectors and subsequent audio frames as described by Kreuk et al. for phoneme boundary detection and classification [3] can inform a model how each MFCC vector is moving over time. Using these four features, I propose a multi-headed CNN, one CNN for each feature, to learn latent features in this data and transform it to a new feature space. The outputs of these new feature spaces can then be concatenated together and passed through a traditional fully connected, feedfoward network for the purpose of classification.

II. Experimental Design

A. Data

I chose to train and test my classifier on the TIMIT Acoustic-Phonetic Continuous Speech Corpus [4], a commonly used dataset for benchmarking automatic speech recognition algorithms. It consists of 630 speakers, from 8 different American dialects, speaking 10 phonetically rich sentences. The corpus contains ∼4 hours of audio at a 16 kHz sample rate and includes annotated audio boundaries and the phoneme represented by the bounded segment. In total there is 61 unique phonemes and a dictionary of 6200 words that can be built from them. However, as noted in [5], there are 15 allophones contained in the dataset, phonemes that don't contribute to any meaning, along with a few other phonemes that are used interchangeably in the dictionary. Therefore, phoneme folding by CMU/MIT standards can reduce the number of required phonemes to classify down to 39 for this dataset.

For my model's input I have chosen 4 feature sets, MFCCs, 1st order MFCC deltas, 2nd order MFCC deltas, and distances of MFCC vectors between subsequent frames. The MFCCs are created by first applying a pre-emphasis filter to the time-domain signal. This finite impulse response filter (FIR) reduces high end frequencies that contain little or no speech energy and reduce the overall noise of the signal. Overlapping frames of this resulting time-domain audio is multiplied by a hamming window function to smooth over the transitions between audio frames. A Short-Time Fourier Transform of the windowed frames is used to sum the frequencies of the audio into bins for each frame of audio. A log function is used to convert the intensities of the frequencies from a linear scale into the decibel scale, which more closely models how humans perceive differences in loudness. These log-power spectrum are converted into Mel filter banks by multiplying each frame by non-linearly sized diamond shape filters, where the low frequencies consist of narrower, linearly increasing sized bands and the higher frequencies are larger, logarithmic increasing width sized bands. This captures more fine grained detail at the lower frequencies where the majority of speech energy resides. It also models human perception, where small variations in low frequencies are more noticeable than larger variations at high frequencies. Finally, MFCCs are obtained by multiplying the Mel filter banks by what is known as a Discrete Cosine Transformation Eq. (1), which smooths out small variations in frequency changes, resulting in more pronounced, decorrelated features. Fig. 1 shows an example of an audio segment containing the phoneme "el", where the edges are zero padded to create a static sized audio length of 3200 samples which represents a time-context of 200 ms at 16 kHz, along with the corresponding MFCCs for the segment. The figure shows the bottom 13 MFCCs, where the majority of speech energy lies. However, the first MFCC contains a constant offset, the blue area, so this coefficient is dropped and only the reamaining 12 MFCCs are used.

$$X_k = \sum_{n=0}^{N-1} x_n cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right] \tag{1}$$

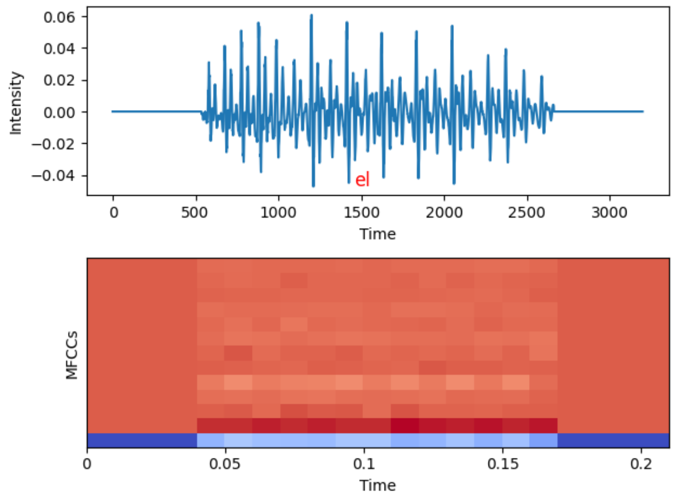

Fig. 1: Raw audio segment of a phoneme and its MFCCs

MFCCs only describe the spectral envelope of an audio segment, however, since phonemes are utterances over time, it's informative for the model to know the dynamics of the audio over the duration of the segment. Therefore, three additional feature sets are derived from the MFCCs. The first order deltas of the MFCCs, which describes the element-wise trajectory of the MFCCs and is modeled using Eq. (2):

$$d_t = \frac{\sum_{n=1}^{N} n(c_{t+n} - c_{t-n})}{2\sum_{n=1}^{N} n^2}, \tag{2}$$

where c_t is the static coeffecient at time-step t, with a typical value for N as 2. Additionally, the 2nd order deltas of the MFCCs to determine the acceleration of the trajectory is calculated using the same equation with the static coefficients c_t replaced by the first order deltas d_t.

Finally, a fourth feature is extracted from the MFCCs which is the distance between subsequent frames of MFCC feature vectors, described in [3] as $D_{t,j} = d(a_{t-j}, a_{t+j})$ representing the Euclidean distance between MFCC feature vectors a_{t-j} and a_{t+j}, where $j \in \{1, 2, 3, 4\}$. This results in 4 additional features per-frame of audio, representing how the MFCC feature vector is changing over time, as opposed to the 1st and 2nd order deltas which represent how each individual coefficient is changing over time.

The Timit dataset consists of 2 parts, training and testing. I created a validation set by randomly selecting 5% of the training set to test the model between training epochs, determining how well the model generalizes, when to anneal the learning rate, and at what point model training should stop to avoid overfitting. The entire test set is used to test the model once training is complete. All three datasets are standardized by feature by subtracting the mean of the training set for that feature and dividing by the standard deviation of the training set for the feature. Thus, the validation and test sets are standardize by the training set statistics since in practice the model will not know the actual distribution of new data, making the validation and test sets a fair approximation of the out-of-sample error.

B. Model

Using four distinct features, each $\in \mathbb{R}^2$, I chose to implement a multi-headed convolutional neural network consisting of 2D convolutional layers, thus treating the features as if they were images. I chose to create a convolutional neural network for each feature so that each model could learn the latent characteristics of each unique feature and transform it into a new feature space. Traditionally, phoneme classification models concatenated the features into a single feature vector, which works well for multi-layer perceptron (MLP) models or recurrent neural networks (RNNs), where there is no inherent spatial locality maintained due to the fully connected properties of those models' layers. For convolutional neural networks, it's more common to concatenate features along the channel axis. However, convolutional filters are applied depth-wise, thus the same learned weights are applied to all channels as they are slid across the input. This means that a model would be optimizing the weights to identify latent features in 4 different feature spaces. I opted for individual models so that the filters in each model are optimized to learn the feature space of a single feature type.

The multi-headed CNN consists of 4 separate convolutional models, one for each feature. Each of these models consist of a series of 5 convolutional blocks. Each convolutional block contains a 2D convolutional layer with 3x3 filters and a stride of 1. The output of the convolutional layer is passed through a batch normalization layer and then a PReLU activation layer, which is then passed through a dropout layer with a dropout probability of 40% for each output. Once all features have been passed through all 5 convolutional blocks, the output of these new feature spaces are flattened and concatenated together and passed through to the classification stage of the model, which resembles a typical MLP. These features are passed through a fully-connected layer containing 500 neurons. The output of this hidden layer is passed through a batch normalization layer and a PReLU activation layer, prior to a dropout layer with a dropout probability of 60% for each output. The signal is then passed to the output layer which consists of 39 neurons, one for each phoneme in the folded Timit library. The output of this layer is passed through a softmax activation, resulting in a probability distribution over the 39 phonemes, where the neuron with the highest probability represents the phoneme with the highest confidence, and thus the chosen prediction given the input features. A representation of this model can be seen in Fig. 2.

For the non-linear activation in the model I chose the parameterized rectified linear unit (PReLU) as described by He et al. [6]. Traditional rectified linear units (ReLU) have shown quick convergence, given a significantly stronger gradient for positive values than Sigmoid and Tanh, however, they can have the side-effect of dead neurons due to the fact that there is

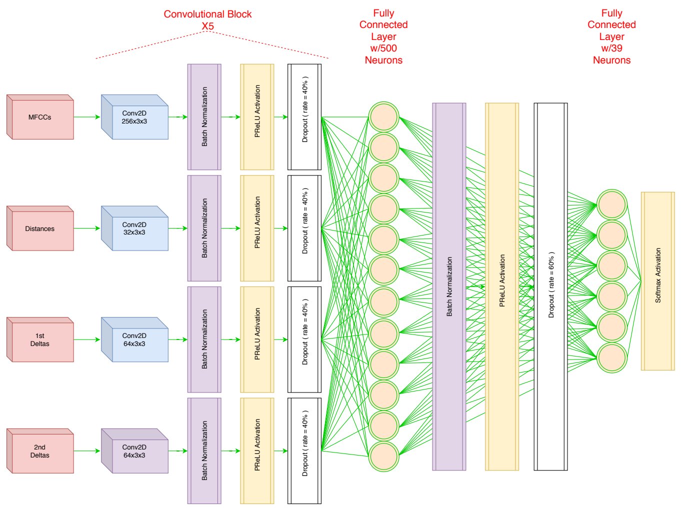

Fig. 2: Multi-Headed CNN for Phoneme Classification

no gradient for non-positive values. A varient of ReLU called Leaky ReLU was developed to alleviate this issue by replacing the $f(y) = 0$ where $y <= 0$ with $f(y) = ay$, as can be seen in Fig. 3. Typically a is 0.1 or 0.2, which creates what is known as a small amount of leak for non-positive values. This still results in a non-linearity with a similar shape as ReLU, but includes a small gradient equal to a for non-positive values, so these neurons can continue to learn even when their input signal is not positive. PReLU is a variant on leaky ReLU, where the amount of leak, a, is a learned parameter. For the model's convolutional layers, this parameter is learned channel wise, so each channel has a different amount of leak. For the PReLU activations before the output layer there is only a single a parameter shared across all the outputs from the previous hidden layer.

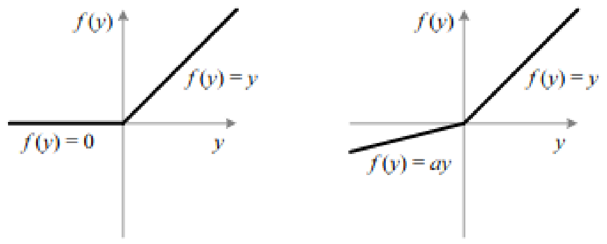

Fig. 3: Comparison between ReLU, left, and LeakyReLU/PReLU, right. The only difference between LeakyReLU and PReLU is that the a value, multiplied to the input for non-positive numbers, is a static parameter for LeakyReLU and a learned parameter for PReLU.

C. Training

For most neural network APIs, Xavier is the standard weight intialization, which samples uniformally proportional to the number of inputs and outputs to the neuron. This assumes a linear activation, which is not the case when using ReLU and PReLU activations. Therefore, for the multi-headed CNN I used Kaiming normal weight initialization, described in [6]. It randomly samples the weights from a normal distribution with a mean of zero and a standard deviation of $\sqrt{2/num\_inputs}$.

The biases are initialized to zero. This keeps the signal from being exponentially amplified after weight initialization.

The model uses the error function Cross Entropy Loss and the AdamW optimizer [7]. This optimizer is a variant of Adam, with the weight decay term decoupled from the adaptive aspect of the algorithm, thus resulting in true L2 regularization. For the weight decay coefficient I used $\lambda = 0.01$ for all layers except batch normalization, PReLU and the final output, which are set to $\lambda = 0.0$, so no regularization occurs in these layers. Batch normalization is trying to correct co-variate shift, so a penalty on these parameters would hinder this. The PReLU activations are learning the amount of leak, which also shouldn't be penalized. And finally, the error is computed at the output layer and thus is absolute and should not be penalized based on model complexity. The AMSGrad [8] option for the optimizer is also enabled, which uses the maximum of the past squared gradient rather than the exponential average to update the parameters.

The model was trained using a mini-batch size of 256, starting with a learning rate of 1e-3. The PyTorch learning rate scheduler Reduce Learning Rate on Plateau is used with a patience of 5 and a factor of 0.1. This reduces the learning rate by a magnitude after the validation loss has not decreased for 5 epochs. Additionally, early stopping is performed with a patience of 10, thus ending training when the validation loss has not decreased for 10 epochs and selecting the model weights from the epoch with the lowest validation loss to be the best model and therefore the one to be tested.

As can be seen in Fig. 4, the model's learning rate reduces after the validation loss plateaus at epoch 28, 36, and 49. The first two reductions help the model settle closer to the nearest local minima. After the final reduction, no additional validation loss decrease occurs and early stopping is initiated after epoch 53. Thus, the final and best model that is used for training is the weights obtained after epoch 43.

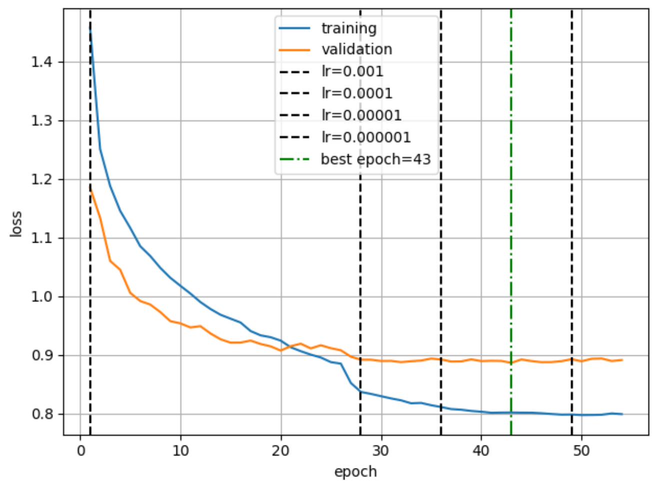

Fig. 4: Training and validation loss for each epoch during the training phase. Dashed lines represent learning rates. Dashed dotted line represents early stopping point where the validation loss was minimized.

III. Results

The results of my experiments can be seen in Table I. Using the original set of phonemes included in the TIMIT dictionary, the multi-headed CNN achieved an ~64% accuracy. With the phoneme folding the model achieved an ~70% accuracy. Current state of the art phoneme classification algorithms achieve a >80% accuracy of the TIMIT test set. However, they are much more computationally complex. Other neural algorithms, as can be seen in the table, preform significantly worse.

TABLE I: Comparison of accuracies on TIMIT test set with other neural algorithms. Results for bi-directional LSTM was found in Hinton *el al.* [9]. MLP, RNN, and LSTM results are from Graves *el al.* [10]. MH-CNN is this work with the multi-headed CNN, both with all phonemes in the dataset and the phoneme folding.

MLP	RNN	LSTM	Bi-LSTM w/CTC loss	MH-CNN	MH-CNN folding
51.4%	64.5%	66.0%	**82.3%**	64.3%	70.4%

IV. Conclusion

By using a multi-headed CNN to learn latent features from four distinct feature sets, the CNN was able to achieve notably better results than other traditional neural approaches such as MLPs and RNNs. However, due to the more than 8 million trainable parameters, the model is prone to overfitting. Therefore, a significantly larger corpus than the TIMIT dataset should produce better results. Future work on this project includes data augmentation, stretching and compressing the audio signals along with applying a variety of filters to increase the size of the training set and help the model generalize better to new data.

V. My Contribution

I certify that all work and committed code to the github repository linked at the top of this page is my own and solely written by me, with one exception. A single function was obtained from the code repository for Kreuk et al. [3], presented with an MIT license, for the extraction of the distance features from the MFCCs. I wrote all functions for data loading, parsing, and transformations. I developed and wrote all code for the multi-headed CNN model. And I wrote all code for training the model. All ideas in this paper are my own original thoughts except where citations exist.

Acknowledgement

I would like to thank professor Meysam Asgari for his instruction and helpful suggestions during this project.

References

[1] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," 2016.
[2] L. Muda, M. Begam, and I. Elamvazuthi, "Voice recognition algorithms using mel frequency cepstral coefficient (mfcc) and dynamic time warping (dtw) techniques," 2010.
[3] F. Kreuk, Y. Sheena, J. Keshet, and Y. Adi, "Phoneme boundary detection using learnable segmental features," 2020.
[4] L. D. Consortium. (1993) TIMIT Acoustic-Phonetic Continuous Speech Corpus. [Online]. Available: https://catalog.ldc.upenn.edu/LDC93S1

[5] K. . Lee and H. . Hon, “Speaker-independent phone recognition using hidden markov models,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 11, pp. 1641–1648, 1989.

[6] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” 2015.

[7] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” 2019.

[8] S. J. Reddi, S. Kale, and S. Kumar, “On the convergence of adam and beyond,” in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=ryQu7f-RZ

[9] A. Graves, A. Mohamed, and G. E. Hinton, “Speech recognition with deep recurrent neural networks,” *CoRR*, vol. abs/1303.5778, 2013. [Online]. Available: http://arxiv.org/abs/1303.5778

[10] A. Graves and J. Schmidhuber, “Framewise phoneme classification with bidirectional lstm and other neural network architectures,” *Neural Networks*, vol. 18, no. 5, pp. 602 – 610, 2005, iJCNN 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608005001206