Wav442Letter: Efficient Speech-to-Text Translation

Andrew Schallwig arschall@umich.edu
Matthew Palazzolo mattbatw@umich.edu
Aditya Singhvi singhvi@umich.edu

Abstract

This paper presents an end-to-end methodology for translating raw audio to text based on the Wav2Letter model [4] while reducing the computational power and training data required. We encode audio using MFCC features and train a 1-dimensional CNN to predict characters for each frame of audio. CTC is used to decode sequences of characters into a probable transcript, which is a human-legible parsing of the predicted characters. We achieved a word-error-rate of 40.1% and a letter-error-rate of 19.1%, competitive with the original Wav2Letter model despite limited resources.

1. Introduction

Speech recognition, or the ability of computers to recognize and interpret human speech, has become an integral part of the way we interact with computers and other technology. Automated speech-to-text technology can improve accessibility for hearing-impaired individuals, enables safety-enhancing technology like voice-texting in vehicles, and increases convenience through voice assistants such as Amazon's Alexa and Google Home. Furthermore, effective speech-to-text technology can enable the creation of realistic, human-like text-to-speech engines.

There are many challenges involved in building a system that can accurately recognize speech, including variations in accent, pronunciation, and background noise. Additionally, the way people speak can vary greatly depending on factors such as their age, gender expression, and native language, making it difficult for a single system to work well for everyone. Overall, speech recognition is a challenging problem because it involves understanding and interpreting spoken language, which is a complex and nuanced form of communication.

However, in recent decades, significant advances have been made in the field of speech recognition as computational power has steadily increased. In tandem, the growing prevalence of consumer technology has expanded the relevance of creating effective speech-to-text methods. In 2016, a team of researchers from Facebook published the Wav2Letter paper [4], using a fully-convolutional network to attain competitive results for the task of speech-to-text. Wav442Letter presents an end-to-end methodology for translating raw audio to text based on the Wav2Letter model while significantly reducing the computational power and volume of training data required.

Our speech-to-text model consists of (1) an acoustic model capable of classifying frames of audio (each spanning about 25 ms) as English characters and (2) a decoder that goes from a sequential representation to a transcript using Connectionist Temporal Classification (CTC). We train on a subset of the open-source LibriSpeech dataset [10] consisting of 2703 clips of speech data, spanning about 5.4 hours. Audio data is encoded using a 13-feature Mel-Frequency Cepstral Coefficients (MFCC) transform.

Our primary contribution to the speech recognition field is to create a simplified model that achieves an acceptable level of accuracy while being much less computationally expensive to train. Wav2Letter trained on 960 hours of data and included around 23 million trainable parameters in the CNN; Wav442Letter is trained on about 0.4% of that dataset and has around 7 million learnable parameters. Ultimately, we aim to provide a practical tool capable of general purpose speech to text translation. We hope to inspire others with our work and advance the field of speech recognition by demonstrating that competitive results are achievable without the computational budget of an industry giant.

2. Related Work

One of the earliest attempts at speech recognition was made by Bell Laboratories in 1952, with a system called the Audrey [9]. The Audrey was able to recognize individual digits from 0 through 9 spoken by a person, a groundbreaking first step in speech recognition. However, it was limited in its ability to understand continuous speech and had a low accuracy rate. Additionally, it was limited in use to only a certain subset of speakers, the developers of the system. Thus, the system was still far from achieving a generalized, language and expression inclusive model.

Over the next several decades, speech recognition technology continued to evolve and improve. In the 1990s, the first commercial speech recognition system, Dragon Dictate

[1] was released. For the first time in history, any ordinary consumer could transcribe audio on their home machine, making speech recognition a widely accessible technology. Although groundbreaking at the time, this rudimentary network lacked the necessary amount of data to properly train.

Google, with its technical prowess and access to significant computational resources, provided the next major step forward for the field. In 2001, Google's Voice Search [8] collected massive amounts of data for matching user queries with human speech. Since then, speech recognition systems have consistently gained performance due to greater access to training data and increases in computational power.

With neural networks gaining popularity in recent years, companies have begun to implement them in modern speech recognition work. In 2017, Google achieved a Word Error Rate (WER) of just 4.9 percent [12], utilizing neural networks. While the exact details of the network are not known, this impressive statistic was likely aided by Google's access to massive datasets and supercomputers.

We decided to implement Facebook's Wav2Letter [4] as it performs comparably to this benchmark, achieving a WER of 7.2% and letter-error-rate (LER) of 6.9%. Furthermore, Wav2Letter is open-source and based on a fully-convolutional system, reducing the computational power required to train and implement it. We aim to balance low training time with high performance by achieving practical results with a simplified pipeline.

3. Methods

Our model can be broken into 2 main parts: phonetic classification using the convolutional network and grapheme (character) alignment using connectionist temporal classification (CTC). The following section outlines our data-processing pipeline, model architecture, and grapheme alignment scheme.

3.1. Dataset

Our model was trained using a subset of the LibriSpeech dataset [10], a corpus of approximately 1000 hours of 16kHz English speech audio samples, along with text transcripts for each audio sample. These samples come from the public domain audiobook and text domain, LibriVox. This dataset provides the model with large amounts of high-quality, clear audio samples with varied speakers and vocabulary, paired with error-free transcripts for ground truth labels. We used the *dev-clean* portion of the dataset, comprised of 2703 of these samples that account for about 5.4 hours of data. 75% of these samples were used for training, while the remainder were used for testing. We opted to use a small, practical portion of the dataset in order to fulfill our goal of recreating Facebook's results using a fraction of their computational power and training time.

3.2. Data Pre-Processing

The original Wav2Letter paper cites three potential inputs to the convolutional network: raw audio waveforms, power-spectrums, and Mel-Frequency Cepstral Coefficients (MFCC). We chose to process the audio as MFCC features, as they had the most promising results in the original paper.

MFCCs are a "spectrum-of-a-spectrum" representation of audio, uniquely suited for machine learning and audio processing applications. As opposed to other Fourier analysis approaches, the use of the Mel scale represents distances between frequencies on a logarithmic scale. This more closely represents human perception of sound, as the human ear can more easily hear differences in frequency at lower frequencies than at higher frequencies [2]. Thus, this is an especially effective representation of how the human ear processes audio information. We chose to use 13 MFCC features, which matches the transform used in the original Wav2Letter paper [4].

The audio dataset from LibriSpeech contains audio samples and transcripts of varying duration, which presents a unique challenge when batching the data for training the model. We opted to batch audio samples of similar duration together to minimize the zero-padding required to align the sizes, which takes the form of long periods of silence for audio samples and spaces for text transcripts. In order to maintain a degree of randomness when passing in batches to our model, we randomized the order of the samples within each batch, as well as randomizing the order of batches when passed into the model. Our data pre-processing pipeline was thus as follows:

1. Split audio samples into test and train sets.
2. Sort audio samples on sample duration.
3. Partition sorted audio samples into batches of 64 samples each.
4. Pad audio samples with silence to the duration of the longest audio sample in each batch, such that all audio samples in a given batch have the same duration. Similarly, pad transcripts to the length of the longest transcript in the batch.
5. Randomize the order of audio samples within batches.
6. Perform MFCC transform on audio samples to generate 13 features for each frame of audio.

This pre-processing pipeline generates data that is batched and organized for training.

3.3. Model Architecture

The architecture for the fully-convolutional phonetic classification model is based directly on the architecture

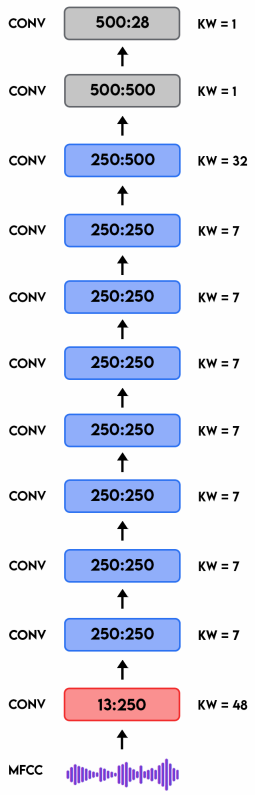

Figure 1. The architecture for the Wav442Letter model. The first layer has stride=2, while all other layers have stride=1. The last two layers act as fully-connected layers with a kernel size of 1.

of the original Wav2Letter paper [4]. This segment of the model is responsible for predicting graphemes, or individual characters, from audio segments. The graphemes in our encoding scheme are defined by the characters present in the LibriSpeech dataset:

- **( )**: Space character. This is represented as a pipe (—) for CTC decoding.
- **(')**: Apostrophe character
- **(A - Z)**: 26 uppercase English letters
- **(-)**: Blank character. This is distinct from the space character. It does not appear in the Librispeech dataset, but is necessary for the CTC decoding scheme.

The architecture of our convolutional neural network (Figure 1) consists of a series of 11 convolutional layers, each followed by a rectified linear unit (ReLU) nonlinearity. No pooling occurs between convolutions; however, the first layer has a large kernel width and a stride of two that achieves a similar effect by creating a large receptive field for future convolutions. Seven identical convolutional layers follow, each with a kernel width of seven and 250 input and output channels each. The ninth layer produces 500 channels (with a kernel width of 32). Finally, the last two layers are effectively fully-connected layers as they have a kernel width of one. The final output of the network are 29 probability logits (each corresponding to a grapheme) for each frame of audio in the input sample.

Our convolutional model differs from Facebook's Wav2Letter in the final three layers, as we use 500 channels for the final portion instead of the original 2000. This creates a 68% reduction in trainable parameters, which significantly decreased training time.

3.4. Grapheme Alignment

The final portion of the model is responsible for translating the output of our CNN architecture to human-readable text. As the CNN simply predicts characters for each time window, these need to be properly grouped and concatenated, such that words are formed, rather than simple strings of characters.

For this task, the system uses the predicted probabilities of characters to reconstruct a transcript using CTC. CTC creates a graph of every possible alignment of letters for a given audio slice, finding the maximal probabilistic path to predict a transcription. This is a departure from the Wav2Letter paper, which implemented a novel AutoSegCriterion (ASC) system for the same task. However, we selected CTC as it is much better understood and documented, and serves the same purpose. We used the *CTCLoss* [6] function available in *PyTorch*'s *nn* library as the loss heuristic while training the model, and *TorchAudio*'s *CTCDecoder* [5] class to reconstruct transcripts while evaluating the final results of the model. The *CTCDecoder* was provided a lexicon of valid words in order to ensure that predicted transcripts fell within the language domain of the LibriSpeech dataset.

4. Experiments

Our final Wav442Letter model achieved a WER of 40.1% and a LER of 19.1% while using 0.38% of the training data and 32% of the trainable parameters of the original Wav2Letter model.

4.1. Training

The model was trained for about 350 epochs on 3.7 hours of data. Several hyperparameters were tested, including the optimizer and learning rate. Stochastic gradient descent (SGD) and Adam optimizers were tested with various parameters. Overall, Adam, when combined with an adaptive learning rate scheduler, worked best for training. An initial learning rate of 10^{-5} was determined to be ideal for our model. Higher learning rates contributed to volatile results while training. Several parameters for the scheduler were also tested, including the patience, decay factor, and threshold. It was determined that these factors did not significantly affect the learning rate.

4.2. Evaluation

Several qualitative and quantitative metrics were used to evaluate the model, including:

- **Word-error-rate (WER):** The WER was calculated as the average proportion of words in a predicted transcript that differed from the ground truth. This metric overstates the error, as a single missed letter marks a word as incorrect.
- **Letter-error-rate (LER):** A normalized Levenshtein distance [7] was used as a proxy for the LER. The Levenshtein distance represents the minimum number of letters that must be inserted, deleted, or substituted to transform a predicted transcript into the ground truth. This distance was normalized by the length of the ground truth transcript for each sample.
- **CTC Loss:** CTC loss was used to train the model, directly calculating the loss between the sequential predictions returned by the model and the ground truth transcript.

4.3. Results

Our results are summarized in Table 1, with Wav2Letter's original results on the right for comparison. We include the number of training parameters and training samples to contextualize the results given our limited computational resources.

	Wav442Letter	**Wav2Letter**[4]
Trainable Parameters	7,486,029	23,282,529
Training Dataset	3.7 hours	960 hours
WER	40.1%	7.2%
LER	19.1%	6.9%

Table 1. A comparison of our results (left) with those of the original Wav2Letter (right). Although our WER and LER are much higher, both the volume of our training data and our computational power were far less than the original.

A sample of results from Wav442Letter are shown in Figure 2. The model achieves fairly accurate results, making mistakes primarily with individual pairs of letters that are phonemically similar. For instance, the [m] in "may" and [b] in "bay," both pronounced bilabially, are confused for each other in Figure 2. However, certain mistakes — such as predicting "paid" for "acid" — seem arbitrary and unjustifiable.

4.4. Future Work

There are several avenues for future work that could improve Wav442Letter, including architecture modifications for regularization and the inclusion of a language model.

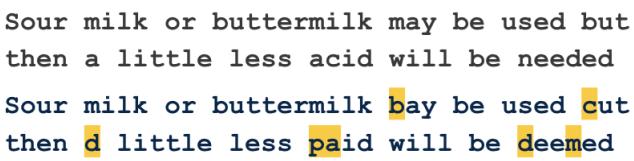

Figure 2. A random, representative sample of our results. The ground truth transcript is shown in gray above, while our prediction is shown in blue below. Errors are highlighted in yellow.

Training on our limited set of training data led to overfitting, as our WER and LER for the train set were much lower. This may account for some of the arbitrary errors seen in the model's predictions, as shown in Figure 2. A possible avenue for increasing generalizability would be adding dropout layers following the final two layers (which effectively act as fully-connected layers). Furthermore, batch normalization layers [11] could be added intermittently between certain convolutional layers. These methods of regularization would help reduce overfitting and increase accuracy on the test set.

Another potential improvement would be incorporating a language model into the CTCDecoder in order to improve our predicted transcripts [3]. A language model would ensure that transcripts are grammatical and match the patterns of spoken and written English. This would enable the model to avoid predicting illogical words such as "d" instead of "a" in Figure 2.

5. Conclusion

Wav442Letter presents a reasonable approximation of Facebook's original Wav2Letter system given a fraction of the computational resources, as demonstrated in the results in Section 4.3. Given the differences in error rates, it's clear that the reduced amount of training data and less nodes in the final layers adversely affected the performance of Wav442Letter when tested on the LibriSpeech dataset. However, we suggest improvements to the original Wav2Letter architecture, including the inclusion of a language model to predict accurate transcripts and adding batch normalization layers to reduce overfitting. Given the opportunity to make these improvements, we believe that the Wav442Letter model could achieve much reduced WER and LER while maintaining our initial goal of creating a practical model on a computational budget.

References

[1] Dragon speech recognition - get more done by voice.

[2] Learning from audio: The mel scale, mel spectrograms, and mel frequency cepstral coefficients, 2021.

[3] Caroline Chen. Asr inference with ctc decoder.

[4] Ronan Collobert, Christian Puhrsch, and Gabriel Synnaeve. Wav2letter: an end-to-end convnet-based speech recognition system. *CoRR*, abs/1609.03193, 2016.

[5] PyTorch Foundation. Ctcdecoder.

[6] PyTorch Foundation. Ctcloss.

[7] V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, Feb. 1966.

[8] Mariella Moon. Google voice search can now handle multiple languages with ease, 2014.

[9] Katia Moskvitch. The machines that learned to listen, 2017.

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

[11] Martin Riva. Batch normalization in convolutional neural networks.

[12] Matt Shannon. Optimizing expected word error rate via sampling for speech recognition. 2017.