Speech to Text Translation Using Google Speech Commands

Amita C. Patil and Hyung Lee
(amita2, hyunglee)@stanford.edu

Abstract

We employed ML methods to recognize spoken words from the *Google Speech Commands* (GSC) dataset. We first converted audio files to Mel Frequency Cepstral Coefficients (MFCC) and used them as features for our models. We identified that a logistic regression model performed poorly with an accuracy of 23% on a ten-word dataset. Hidden Markov Model (HMM) with Gaussian Mixture Model (GMM) achieved an accuracy of 65% on the same set, and the accuracy increased to 83% when we trained a simpler HMM on our own voice. The best performing model was Convolutional Neural Network (CNN) based model which achieved nearly 90% accuracy.

1 Motivation

Most voice recognition devices rely on keyword detection to start interaction (e.g. "Ok Google" or "Hey Siri"). Once the device detects the keyword, it can send audio to a cloud based service. The initial detection of word commands is impractical to run on the cloud since it would require sending audio data over the web all the time, which would be costly and could compromise privacy. On the other hand, a model running on the device has hard resource constraints and requires fewer calculations than their cloud counter-parts. To extend battery life, computations must also be energy efficient. Since users expect a fast response from their devices, and since network latency is highly dependent on the external factors, having local acknowledgement that a command was received is critical for good user experience.

2 Related Work

For your project we use *version 0.02 (V2)* of Google Speech Commands (GSC) dataset [11]. Google trained their default CNN for Small-footprint Keyword Spotting [5]; their results on two versions of the dataset are summarized in Table 1. Before popularity of deep neural

Data	V1 Training	V2 Training
V1 Test	85.4%	89.7%
V2 Test	82.7%	88.2%

Table 1: Google's top-One accuracy evaluations on two versions of their dataset

networks (DNN), Hidden Markov Models (HMM) were predominantly used for speech recognition [2]. Most current speech recognition systems also use HMMs to deal with the temporal variability of speech and Gaussian mixture models (GMM) to determine how well each HMM state fits a frame or a short window of frames of coefficients that represents the acoustic input [3]. NN have become increasingly popular over past two decades [4]. Using CNN's for speech recognition is a relatively recent phenomenon. CNN model trained on the GSC achieves greater than 88% accuracy, and other CNN based models that are paired with LSTMs have achieved greater than 95% accuracy [8].

3 Dataset and Features

3.1 Dataset

We used *version 0.02* of the *GSC* [6], which contains 105,829 audio recordings of people speaking a single word per WAV file. There are 20 core words that most speakers recorded five times each: "zero", "one", to "nine", "forward", "backward", "up", "down", "yes", "no",

Word	Number of Utterances
Zero	4,052
...	...
Backward	1,664
...	...
Bed	2,014
...	...

Table 2: How many recordings of each word are present in the dataset

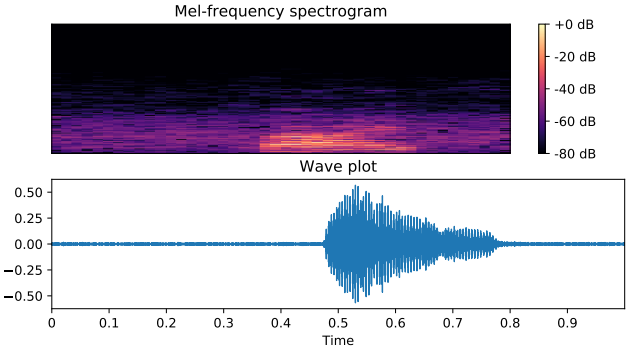


Figure 1: MFCC feature vector and wave plot of a person saying "Five"

"on", "off", "right", "left", "go" and "stop". There are also auxiliary words, which most speakers only said once: "bed", "bird", "cat", "dog", "happy", "house", "marvin", "sheila", "tree", "follow", "learn", "visual" and "wow". In addition to these, there were also files that contain background noise.
The dataset provides a list of recordings to be used for validation and testing, which we used to generate train/validation/test sets. The audio files have a minimum size ranging from 7K to 32K, when the file contains a spoken word, whereas they could be as large as 3M when the file contains background noise. CNN models handled this variable dimension with padding; whereas HMM inherently can handle varying time-series data.

3.2 Features

The first step in any automatic speech recognition system is to extract features that are good for identifying the linguistic content and discard extraneous information like background noise. MFCCs, introduced by Davis et al. in the 1980's, are widely used in speech recognition and have been the state-of-the-art ever since [13]. The mel-frequency cepstrum (MFC) represents the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency. In MFC, the frequency bands are equally spaced on the mel scale, which approximates the human auditory system's response more closely than the linearly-spaced frequency bands. We first convert audio files to MFCC features by using the librosa API [14], and then use these two-dimensional features to train our models (Figure 1). The two dimensions represent spectral content and time.

4 Metrics

The most straightforward evaluation metric calculate how many utterances our model identified correctly (top-one error). Since we are dealing with multiple classes, we also calculate weighted accuracy, precision, recall, F1-score and confusion matrix. For a baseline we use Google's provided model that achieves top-one error of 88.2% on V2 test set. Accuracy of a human performing the task could be used for an Oracle.

5 Models

5.1 Logistic Regression

Our Logistic Regression model (drawn from an online resource[16]) has much lower accuracy compared to either HMM or CNN models. We believe this is because, in order to use regression on the dataset, we had to flatten the inputs into one-dimensional vectors. This involved either simply flattening the two-dimensional MFCC matrix into a one-dimensional vector or averaging the value of each MFCC component per time frame. This can cause serious loss of temporal/spectral content in speech. We believe this explains the relatively low 33.2% accuracy on the test set.

MFCC	Time	Iterations	Test Set Accuracy
20	11	20000	33.20%
40	11	35000	31.90%
40	11	50000	31.20%
40	11	100000	30.70%
20	11	5000	25.10%

5.2 HMM

We use HMM with Mixture of Gaussians to classify 20 core command words. In [1], L. Rabiner provides a thorough and methodical review of the theoretical aspects of HMM for speech recognition. The HMM model can be characterized by:
1. *N* individual states; denoted as $S = \{S_1, S_2, ..., S_N\}$. State at time *t* is denoted as q_t
2. *M* distinct observation symbols per state; denoted as $v = \{v_1, v_2, ..., v_M\}$
3. State transition probability distribution $A = \{a_{ij}\}$; where $\{a_{ij}\} = P[q_{t+1} = S_j | q_t = S_i]$, $1 \leq i, j \leq N$
4. Observation symbol probability distribution in state *j*; denoted as $B = \{b_j(k)\}$
where $\{b_j(k)\} = P[v_k \text{ at } t | q_t = S_j]$, $1 \leq j \leq N$ and $1 \leq k \leq M$.
For our model the observable symbols are drawn from a pdf representing Mixture of Gaussians (GMM).
5. The initial state distribution $\pi = \{\pi_i\}$; where $\{\pi_i\} = P[q_1 = S_i]\}$, $1 \leq i \leq N$

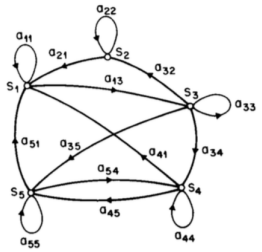


Figure 2: Illustration of HMM with 5 states (labeled S_1 to S_5)

A complete specification of an HMM requires specification of two model parameters (*N* and *M*), and three probability measures *A*, *B*, and π. For compact notation we use $\lambda = (A, B, \pi)$ to indicate complete parameter set of the model.
There are three aspects of HMM model:
1. Given observation sequence $O = O_1 O_2 ... O_T$ and model λ, compute $P[O|\lambda]$. This can be viewed as scoring how well a given model matches a given observation sequence.
2. Given observation sequence $O = O_1 O_2 ... O_T$ and model λ, compute $P[O|\lambda]$. i.e. how to choose corresponding sequence $Q = q_1 q_2 ... q_T$ that best explains the observations. The intuition here is to uncover the hidden part of the model, i.e. to find *"correct"* state sequence using some optimality criteria, and
3. Finally the most critical step of training the model. This is done using an iterative EM (expectation-maximization) procedure that find parameters $\lambda = (A, B, \pi)$ so as to maximize $P[O|\lambda]$.

5.3 CNN

Our most successful CNN model was based on the Keras example model for training on the MNIST image dataset[9]. The input was shaped into: n examples $\times$ MFCC components $\times$ time $\times$ depth = 1. Depth of 1 was used to accommodate the CNN model, typically used for image recognition. The model consists of:
2D convolution: 32 filters, kernel size 3×3, RELU activation
2D convolution: 64 filters, kernel size 3×3, RELU activation
2D Max pooling: pool size 2×2
Fully connected layer: 128 neurons, dropout of 0.25, RELU activation
Final fully connected layer: # of classes, dropout of 0.5, SOFTMAX activation
We used categorical cross-entropy loss:

$$CE = -\sum_{i=1}^{C} y_i log(f(s)_i); f(s)_i = \frac{e^{s_i}}{\sum_{j=1}^{C} e^{s_j}}$$

Here *C* denotes # of distinct classes in our Multi-Class classification, y_i is one-hot representation of our ground truth labels and *s* is input to softmax layer. In both 2D convolution layers, we used *keras* default value for stride=1 and padding. By using kernels, the convolution layer

makes the input easier to process (fewer model parameters) without losing essential features [10]. Passing the kernel over the input results in a feature map that is given by the function[12]:

$$G[m,n] = (f * h)[m,n] = \sum_{j=1} \sum_{k=1} h[j,k] f[m-j, n-k]$$

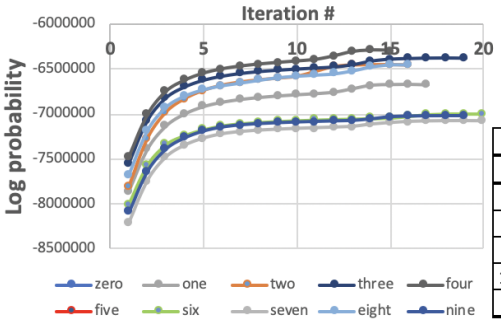
The input is f and the kernel is h. The pooling layer reduce the spatial size of the convolved features, meaning the model performs fewer computations.[10]. It selects a maximum value from each region and put it in the corresponding place in the output.[12]. Finally, we also experimented with a similar model albeit with one convolution and smaller kernel sizes. [7].

6 Results and Analysis

6.1 HMM

We leverage hmmLearn API [15] for HMM models and train first on 3, 5 and finally on 10 command words. For hyper-parameter tuning we varied # of states (N) in the model, # of MFCC features (M), and # of GMM mixtures to get high accuracy on validation set. We trained the model for varied the # of iterations and/or until the loss for all models converged (Figure 3a). Note EM algorithm used to train HMM is prone to get stuck at a local optima, we used random seed to overcome this.

We observed that merely increasing (N) and MFCC features (M) did not improve accuracy (Figure 3b). We believe this is because GMMs have a serious shortcoming – they are statistically inefficient for modeling data that lie on or near a non-linear manifold. For eg., modeling set of points that lie very close to the surface of a sphere only requires a few parameters using an appropriate model class, but it requires a very large number of diagonal Gaussians or a fairly large number of full-covariance Gaussians. True underlying structure of speech produced by modulating a relatively small number of parameters of a dynamical system is much lower-dimensional than is immediately apparent in a window that contains hundreds of MFCCs . We believe, therefore, that other types of models that exploit information in large window of frames may work better than GMMs.
Our HMM models have higher bias compared to CNN baseline model from Google. Precision/Recall was more or less balanced across all classes expect label *"one"* which noticeably had lower recall compared to other classes. Interestingly, we also observe models trained on smaller subset of # of classes or on fewer distinct people voices perform better (Figure 3b). We also noticed that when we trained a simpler HMM using recordings of our voice it performed much better with a 83.3% test accuracy. We believe the increased accuracy for this model (as well as models trained on fewer classes) is due to the fact that the simple HMM models are not able to generalize to different people's voices as well as it can perform on a handful of distinct voices.



(a)

Our "optimized" HMM models					# of examples			Accuracy (%)		
# of classes	N	M	# of GMM mixtures	# of iterations	Train	Validation	Test	Train	Validation	Test
3	40	13	9	30	4881	612	604	81.0	77.1	75.3
5	50	13	9	30	12683	1482	1652	81.1	77.5	76.1
10 (numbers)	100	13	9	30	25832	3643	4107	77.4	66.7	65.8
10 (non-numbers)	100	13	9	30	21852	2565	2791	74.9	66.6	64.6
10 (our voice)	3	6	2	20	130	30		90.8	83.3	

(b)

Figure 3: (a) Log probability of loss for 10 HMM models, and (b) Summary of hyper-parameters tuning over # of model classes

6.2 CNN

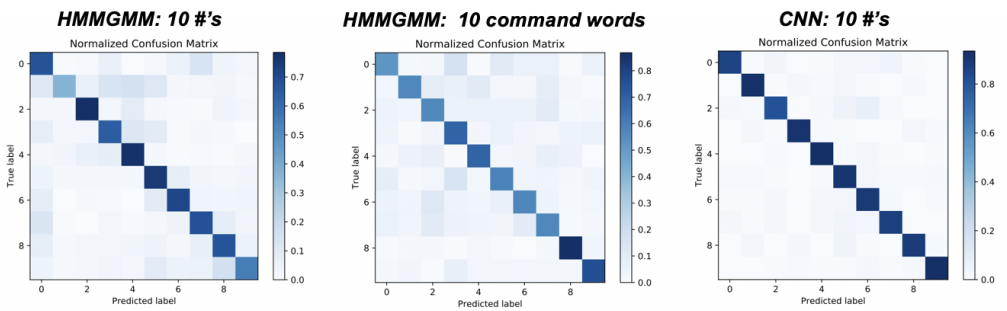
We examined CNN performance (for model with 2D conv) on several small datasets to see if performance differed based on word choice.

Dataset	Test Set Accuracy
Bed,Cat,Happy	92.79%
Bird,Five,Nine,Off,Sheila	93.68%
Go,No,Wow	88.89%
Five,Follow,Forward	91.79%
Bed,Cat,Happy,Bird,Five	91.36%

We observed a result that confused us as first: our model performed better on the five-word set (row 2) compared to three-word set (row 1). However, testing on a different 3-word set with words that had similar vowel sounds (row 3) showed that as words sound more similar, performance degraded. The second five-word set (row 5) contains the same words as the three-word set with similar sounding words (row 1) plus two more words. The model fared worse on this set than on the first five-word set (row 3), showing more support for our hypothesis. On the same ten-word dataset, the model with 2D convolution outperformed the model with 1D convolutional layer: 89.56% test set accuracy vs. 84.44%. This makes sense; since 2D convolutions should be able to distill the spectral and temporal features from the input better. In terms of hyperparameter tuning, we experimented with changing the kernel size. On the same three-word dataset, the model with two convolutional layers and a kernel of size 2x2 achieved 90.37% accuracy, versus 92.21% for a kernel of size 3x3. This shows that a smaller kernel size resulted in a feature map that was less representative of the original input. This may suggest that the best kernel size is one that can decompose the input into recognizable constituent parts without becoming too granular. Using a 4x4 kernel resulted in accuracy of 88.49%, suggesting that a kernel that is too large can hamper performance as well. In general, Precision/Recall/F1-score were more or less balanced across all classes (Figure4).

6.3 Comparing CNN with HMM

Finally we compare our CNN and HMM models. *"Confusion Matrices"* are a nice way to visual how our different models are doing over all classes. Note that the diagonal entries in confusion matrix correspond to correctly classified examples whereas off diagonal entries are ones that our model failed to identify correctly. It is evident that CNN model performs better over most classes (Figure 4). We believe this is due to high expressive nature of CNN models with relatively few parameters. Figure 5 provides a summary of all our models relative to Google's CNN model.



SUMMARY OF TEST SET RESULTS FOR CNN MODEL TRAINED ON #s

label	precision	recall	f1-score	# of examples
one	0.93	0.86	0.9	775
two	0.92	0.93	0.92	797
three	0.88	0.81	0.85	734
four	0.95	0.92	0.93	806
five	0.91	0.93	0.92	770
six	0.8	0.91	0.85	785
seven	0.81	0.9	0.85	782
eight	0.9	0.87	0.89	748
nine	0.93	0.89	0.91	831
zero	0.95	0.94	0.94	754
Accuracy			0.9	7782
Macro Avg.	0.9	0.9	0.9	7782
Weighted Avg.	0.9	0.9	0.9	7782

Figure 4: Confusion matrices for HMM models trained on #'s and command words and CNN model trained on #'s

Algorithm	Dataset	Training Accuracy	Test Accuracy	Test Precision (weighted)	Test Recall (weighted)	Test F1-score (weighted)	Test # of test examples
Multinomial LR	GSC v2 - numbers 0 to 9		0.23	0.40	0.17	0.23	7782.0
HMM-GMM	GSC v2 - numbers 0 to 9	0.77	0.66	0.67	0.66	0.65	4107
HMM-GMM	GSC v2 - command words	0.75	0.65	0.65	0.65	0.65	2791
HMM-GMM	Our voice - numbers 0 to 9	0.91	0.83	0.80	0.83	0.79	30
CNN	GSC v2 - numbers 0 to 9	0.99	0.90	0.90	0.90	0.90	7782
CNN	GSC v2 - command words	0.97	0.90	0.90	0.89	0.90	5542
Google's CNN	GSC v2 - command+auxillary words		0.84				

Figure 5: Performance summary of our models on Google Speech Commands

7 Conclusion

Both our HMM and CNN produced models were able to achieve reasonable accuracy on a limited number of distinct speakers and classes. However, for improved accuracy over entire *GSC* dataset, we likely need more sophisticated methods that either combine Neural Networks to classify individual frames of acoustic data and use their output distributions as emission probabilities for a HMM; or alternatively end-to-end deep bidirectional LSTM recurrent neural network architectures.

8 Github Link

`https://github.com/hyungl/cs229fall2019/tree/master/Project/`
In this repository we have a folder that contains a small subset of *WAV* files that our model classified correctly/incorrectly.

9 Contributions

Hyung Lee: Research, model training, data generation, and error analysis for logistic regression and convolutional neural network models; writing portions of milestone, poster, and final paper
Amita Patil contributed to feature extraction, training and analysis for HMM models and writing final paper, poster, milestone and proposal.

References

[1] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[2] B. H. Juang and L. R. Rabiner, "Hidden markov models for speech recognition," *Technometrics*, vol. 33, no. 3, pp. 251–272, 1991.

[3] M. J. Gales, "Maximum likelihood linear transformations for hmm-based speech recognition," *Computer speech & language*, vol. 12, no. 2, pp. 75–98, 1998.

[4] O. A.-H. et al, "Convolutional neural networks for speech recognition," *IEEE/ACM TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING*, vol. 22, 2014. [Online]. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/CNN_ASLPTrans2-14.pdf.

[5] T. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," 2015.

[6] (Aug. 2017). Google ai blog: Launching the speech commands dataset, [Online]. Available: https://ai.googleblog.com/2017/08/launching-speech-commands-dataset.html.

[7] M. K. Mandal, *Building a Dead Simple Speech Recognition Engine using ConvNet in Keras*, Nov. 2017. [Online]. Available: https://medium.com/manash-en-blog/building-a-dead-simple-word-recognition-engine-using-convnet-in-keras-25e72c19c12b.

[8] D. Andrade, S. Leo, M. Viana, and C. Bernkopf, "A neural attention model for speech command recognition," Aug. 2018.

[9] F. Chollet, *simple convnet on the MNIST dataset*, Feb. 2018. [Online]. Available: https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py.

[10] S. Saha, *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*, Dec. 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[11] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *CoRR*, vol. abs/1804.03209, 2018. arXiv: 1804.03209. [Online]. Available: http://arxiv.org/abs/1804.03209.

[12] P. Skalski, *Gentle Dive into Math Behind Convolutional Neural Networks*, Apr. 2019. [Online]. Available: https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9.

[13] . [Online]. Available: https://en.wikipedia.org/wiki/Mel-frequency_cepstrum.

[14] . [Online]. Available: https://librosa.github.io/librosa/tutorial.html.

[15] . [Online]. Available: https://hmmlearn.readthedocs.io/en/latest/tutorial.html.

[16] *ML | Logistic Regression using Tensorflow*. [Online]. Available: https://www.geeksforgeeks.org/ml-logistic-regression-using-tensorflow/.