Misha Dubuc & Dhaval Patel
COMSC 415
Prof S. Cates
14 May 2019

Speech Recognition

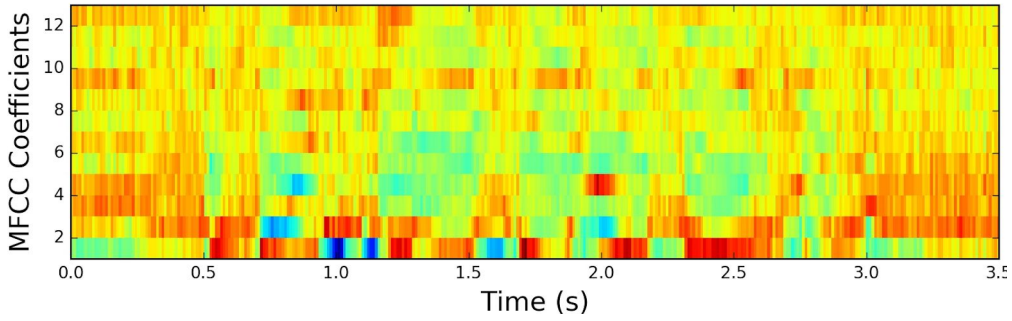
Introduction

The purpose of this project was to implement and analyze a basic speech recognizer. For the training data, Kaggle provided thirty-one folders, each including several .WAV recordings of an assigned word (the only exception being the 'background noise' folder, which only consists of six recordings). Only twelve words and categories were of importance ("yes," "no," "up," "down," "left," "right," "on," "off," "stop," "go," unknown words, and silence), as these were the sounds the model had to recognize.

We noticed early in our research that the test files were not labeled. However, we soon learned that Kaggle expected its competitors to split the training data into training, validation, and test sets; Kaggle ran submitted models on the test data to determine the leaderboards. Therefore, we split the training data into training (60%), validation (20%), and testing (20%) sets, which we had not normally done before. This ratio is encouraged by machine learning specialist and Ian Goodfellow supervisor Andrew Ng (Shrott).

Data Preprocessing

To analyze the data, the audio files needed to be converted into **Mel-Frequency Cepstral Coefficients** (**MFCC**), a representation of sound features that allow audio files to be comparable with each other. Beth Logan of Compaq Computer Corporation provides a good explanation of this complex transformation.



Visual representation of an MMFC (Fayek)

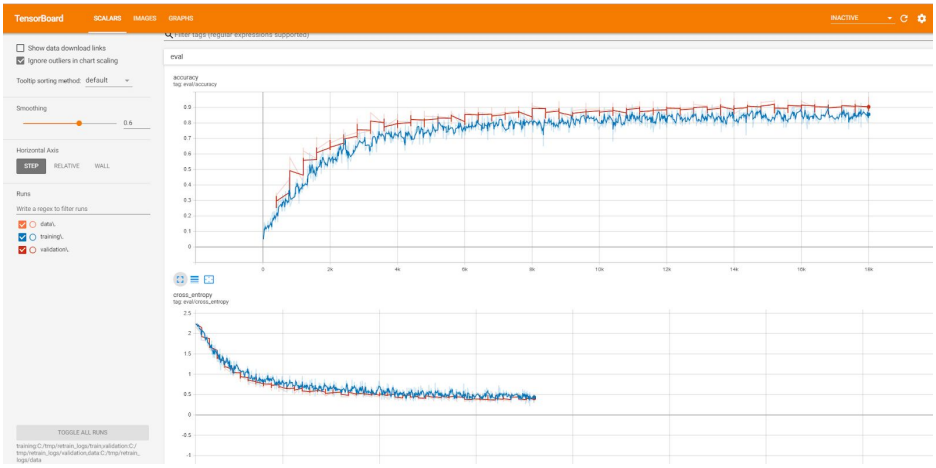
Firstly, audio files are separated into various frames, each being milliseconds long. Each frame is then processed by a Discrete Fourier Transform, which extracts their frequencies and amplitudes. The logarithm of the amplitude is taken, as studies have shown that volume takes on a logarithmic nature.

The spectrum is smoothed so important frequencies are made more evident. Lower frequencies have shown to be more impactful in speech recognition, so the binning of frequencies is based on this 'Mel frequency scale.' Finally, a form of Principal Components Analysis called Discrete Cosine Transform reduces the dimensionality of each frame.

Ultimately, for our preprocessing program (provided by Manash Mandal), all MFCCs of a certain word are vectorized, and those vectors are stored in a word-specific numpy file that the model can analyze.

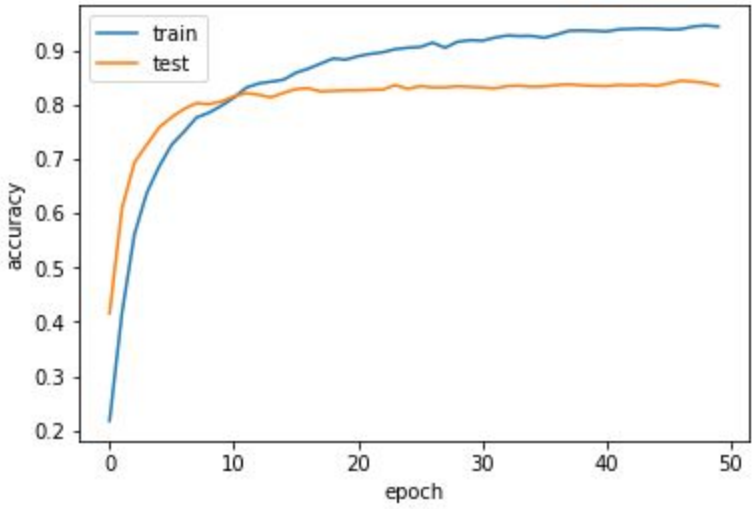
Training Model

We originally investigated an example model provided by TensorFlow on their GitHub repository. Running it gives a test accuracy of 87.5% (they split the training data into 80% training, 10% validation, and 10% testing); this is quite good, but this score makes sense for their example model, as Kaggle competitors were expected to reach at least 90%. Additionally, data visualizer TensorBoard provided the following accuracy trends, where the blue and red represent the training and validation sets, respectively:



The dense structure of TensorFlow's code, however, made it difficult to discern where and how to adjust the parameters. Therefore, we settled on Mandal's model as a foundation, a simple convolutional neural network consisting of three 2D Convolutional layers, one Max Pooling layer, three Dropout layers, two ReLU activations, and one final Softmax activation.

After splitting the training data into training and validation sets, we ran the unchanged model on the data:



As expected, the training accuracy was quite high, with a final accuracy of 94.12%. The validation accuracy itself was not terrible, with a final accuracy of 83.47%. However, the graph highlights not only the near 11% discrepancy between the training and validation accuracies, but also the stagnation of the validation accuracy, as it did not see any major growth by the tenth epoch. This told us that while the model worked decently as it was, there was some overfitting that begged for resolving.

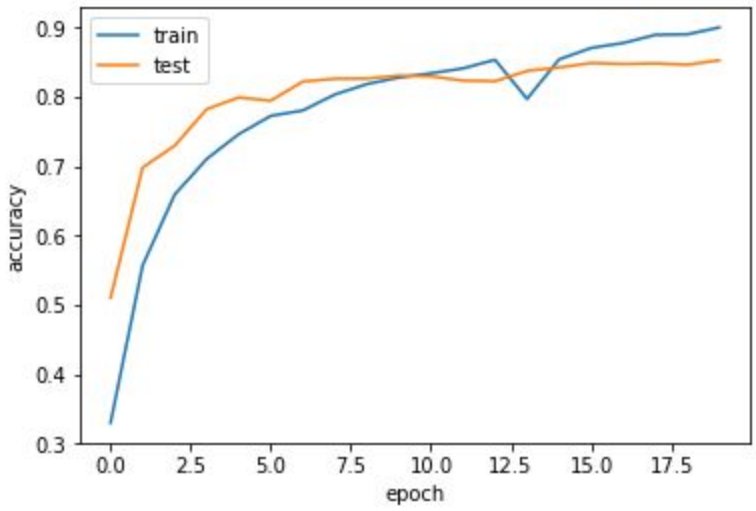
Model Revisement

Because the overfitting was not egregious, we did not want to fight it too aggressively. We briefly considered changing the structure of the neural network from Convolutional to Recurrent, as RNNs are used frequently for speech recognition. However, we realized that its backpropagation-through-time feature is only practical for audio that lasts at least for full sentences, not just for one word.

After testing various configurations, the following changes were determined to work best:

Parameter	Change	Reasoning
Epochs	Decreased from 50 to 20	Prevents excess training. Not necessary when validation accuracy plateaus early.
Batch size	Decreased from 100 to 64	Increases generality, though there is no consensus on why (Mudigere, et. al).
Dropout	Increased two from 0.25 to 0.30	Helps fight overfitting by hiding random units, so the model does not mimic the training set. Encouraged by Ian Goodfellow.
Batch Normalization	Added one layer	Normalizes and regularizes its input by providing some noise (Doukkali).

We decided to leave the kernel sizes of the convolutional layers alone, as they seemed small enough.



The training accuracy decreased by 3.70% to 90.42%, but this was expected, as we actively tried to make the model more general. The improvement of the validation accuracy was not massive; it reached 85.29%, a difference of only 1.82%. Nevertheless, with the techniques we know how to use, it was our highest accuracy score achieved.

Testing Results

Running the model on the testing set provided the following confusion matrix:

```
Test Error
Correct classification rate (Neural Network):
0.8497188618163231
Confusion matrix (Neural Network):
[[356  15   2  12   0   4   0  10  16   3   6]
 [  8 377   2  46   8   1   1   4  22   3   5]
 [  1   0 391   4   1   0  10   3  16   5  23]
 [  8  39   3 435   1   1   1   2   9   2   9]
 [  1   0   4   2 391  23   0   6  14  29   0]
 [  2   3   1   2  13 451   3   2  33   6   0]
 [  1   0  17   1   1   5 428   1  28   4   1]
 [  5  15   6   2   2   1   0 381  19   3   3]
 [ 22  27  15  16   4  32  18  17 940  24  24]
 [  1   3   1   3  20   8   1  18  25 393   0]
 [  2   0  11   8   0   1   2   0  14   0 444]]
```

The model produced a testing accuracy of 84.97%, a mere 0.32% dip from the validation accuracy. In general, the number of misclassified words per category is rather low, considering there were thousands of words; the main diagonal shows consistently good results. However, while not extremely so, the average misclassification counts across the ‘unknown’ row and column is higher than for all other categories. This may be attributed to phonetic similarities (as a contrived example, the unknown word “nine” containing the same long vowel as “right”).

Kaggle Submission

After our work, we were curious to see how a Kaggle user may have approached the competition. In particular, we reviewed Pavel Ostyakov’s team’s submission, which won fourth place.

Ostyakov’s team employed ensembling by using elements from VGG, ResNet (both powerful variations of the CNN), and various RNNs. VGG uses a specific configuration of max pooling, fully-connected, and Softmax layers, as well as 3-by-3 convolutional layers that are stacked on top of each other in increasing depth (Rosebrock). ResNets, or Residual Networks, use skip connection, a method of skipping certain layers to mitigate the vanishing gradient problem (Dwivedi).

Ostyakov’s team also implemented various methods of preprocessing the audio files. For example, they worked with a tempogram, which measures the specific speed of speech in each frame. This can be used to discern certain patterns in phonetics, such as stressed syllables that may be elongated (Music Information Retrieval).

By himself, Ostyakov reached eleventh place, but merging models with new collaborators led to a fourth-place win.

References

Doukkali, F. (2017, October 20). Batch normalization in neural networks. Retrieved from https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c

Dwivedi, P. (2019, January 4). Understanding and coding a ResNet in Keras. Retrieved from https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33

Fayek, H. (2016). *MFCCs*. Retrieved from https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html

Goodfellow, I. (2016, July 19). *Practical methodology for deploying machine learning* [Video file]. Retrieved from https://www.youtube.com/watch?v=NKiwFF_zBu4&t=1287s

Kaggle TensorFlow speech recognition challenge. (2018, February 23). Retrieved from https://dinantdatascientist.blogspot.com/2018/02/kaggle-tensorflow-speech-recognition.html

Kaggle. (2018). TensorFlow speech recognition challenge. Retrieved from https://www.kaggle.com/c/tensorflow-speech-recognition-challenge

Logan, B. (n.d.). Mel frequency cepstral coefficients for music modeling. Retrieved from http://musicweb.ucsd.edu/~sdubnov/CATbox/Reader/logan00mel.pdf

Mandal, M. K. (2017, November 21). Building a dead simple speech recognition engine using ConvNet in Keras. Retrieved from https://blog.manash.me/building-a-dead-simple-word-recognition-engine-using-convnet-in-keras-25e72c19c12b

Mudigere, D., Keskar, N. S., Nocedal, J., Smelyanskiy, M., & Tang, P. (2017). On large-batch training for deep learning: Generalization gap and sharp minima. Retrieved from http://users.iems.northwestern.edu/~nitish/Research.html

Ostyakov, P. (2018). Our approach [4th place]. Retrieved from https://www.kaggle.com/c/tensorflow-speech-recognition-challenge/discussion/47674#latest-427038

Rosebrock, A. (2017, March 20). ImageNet: VGGNet, ResNet, Inception, and Xception with Keras. Retrieved from https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/

Shrott, R. (2017, October 25). Deep learning specialization by Andrew Ng: 21 lessons learned. Retrieved from https://towardsdatascience.com/deep-learning-specialization-by-andrew-ng-21-lessons-learned-15ffaaef627c

Tempo estimation. (n.d.). Retrieved from https://musicinformationretrieval.com/tempo_estimation.html