Tyler DeBruin
Nolan Angerbauer

4/30/2024

Final Project - Command Word Speech Recognition

Abstract

Speech Recognition is presented as the core challenge that we look to solve. In this report, we source a data set from (Warden et al. 2018) that features 1 second audio clips tied to a label. Those labels feature 1 of 35 command words, and silence. We showcase downloading the data in an efficient manner, training a Convolutional Neural Network on the streamed data, and performing data cartography.

Introduction:

Modern Internet-Of-Thing devices are heavily dependent on command words that activate the device. These devices contain hardware that is specialized in listening, and inferring that word to activate. All of that takes place on the local device to ensure the security and safety of those that own such a device. The most common ones known today are Google Home, or Alexa kiosks that serve as a one stop shop for home assistance. This report sets out on training a neural network on speech recognition across many different command words, in an attempt to expand upon this. The applications of a local device being able to perform inference without a network connection are pivotal to the privacy of users; and users will only feel comfortable knowing there isn't a chance that their personal audio can be recorded without their consent.

We began development by constructing a containerized application for easy use and replication of this report on any hardware. The container houses several key pieces: A Jupyter Notebook server, a PostgreSql server, and an Apache Spark instance that is able to communicate between both with Pyspark. We then pull in the (Warden et al. 2018) dataset from Hugginface.co by leveraging the datasets library. That data is streamed into our notebook utilizing Apache Arrow, which the HuggingFace library already provides. We begin by curating the data; padding and shortening all audio files to a fixed width of 16kHz. This gives every sample audio file a total of 1 second of time. We then perform a Wiener filter across the audio file to reduce the noise. Once each data sample is processed, we train all samples with a Convolutional Neural Network to yield a model that can provide inference with Top-One accuracy of 73% across 36 different labels. To correct this imbalance, we also reference Data Cartography (Swayamdipta et al., 2020) to attempt to clean, and retrain the model for a higher Top-One accuracy.

Dataset:

The Speech Command Dataset (Warden et al. 2018) was a crowd sourced attempt at building a dataset that could help train machine learning models. The audio files appearing within the dataset have been anonymized; they were selected based on a few characteristics that Warden thought would make a good audio dataset. Warden wanted to only include English words; short syllable words, and randomized the collection across many different accents making it easy for a model to generalize. Lastly, (Warden et al. 2018) looked to constrain the input to a single second - which isn't restrictive given the basis for this dataset.

Figure 1: Class Distributions

Word	Count	Word	Count	Word	Count
Backward	1664	Happy	2054	Sheila	2022
Bed	2014	House	2113	Six	3860
Bird	2064	Learn	1575	Stop	3872
Cat	2031	Left	3801	Three	3727
Dog	2128	Marvin	2100	Tree	1759
Down	3917	Nine	3934	Two	3880
Eight	3787	No	3941	Up	3723
Five	4052	Off	3745	Visual	1592
Follow	1579	On	3845	Wow	2123
Forward	1557	One	3890	Yes	4044
Four	3728	Right	3778		
Go	3880	Seven	3998		

Speech Command Dataset (Warden et al. 2018) features a total of 36 classes; 35 total command words, and Unknown/Silence. Figure 1 showcases the distributions of these classes. Some words are featured in this dataset at an imbalanced rate, and he has also flagged some of these as non-auxiliary words. The intent is to teach the model to differentiate between command words, and other speech. Warden also includes computer generated static, and silence as examples within the dataset.

There are a total of 2,618 unique speakers across the dataset.

Each record in the dataset features an audio file, including a file path. The filepath acts as the key for that unique record. The audio files are in a wav format, sampled at 16kHZ each. There are rows in the dataset that contain more or less samples - which means that the audio file was recorded for milliseconds more or less than the average. Each record has a golden

label, a speaker id, whether the word is an auxiliary word or not, and the utterance Id indicating how many times this specific speaker has given this word as an example.

```
{
'file': 'backward/2356b88d_nohash_0.wav',
 'audio': {'path': '2356b88d_nohash_0.wav',
  'array': tensor([ 0.0000,  0.0000,  0.0000,  ..., -0.0001, -0.0002, -0.0001]),
  'sampling_rate': tensor(16000)},
 'label': tensor(30),
 'is_unknown': tensor(True),
 'speaker_id': '2356b88d',
 'utterance_id': tensor(0)
}
```

Figure 2: Test/Train/Validation Split

Train	Validation	Test
84848	9982	4890

In total, there are 99,720 data points in this dataset. In our project, we utilized a 85%, 15% training split. The exact counts are featured in Figure 2.

Related Works:

Existing audio datasets include:

Librispeech (Chen et al. 2015) - This dataset features 1000 hours of human speech based on audio books. The audio files are sampled at 16kHz, a standard for audio of human speech.

Common Voice [(Ardila et al. 2020)] - 9,283 recorded hours of speech; including demographic metadata such as age, sex, and accent. Also includes 7,335 validated hours in 60 languages.

Design

Our solution runs on a Docker container for easy accessibility. In this container we download the needed tools such as PyTorch, Postgres, Jupyter Notebooks, and Apache Spark. We had original plans to download the whole dataset into the container but we opted to stream the data using Apache Arrow, which is built into the datasets library provided by Huggingface.co.

The initial setup of our application requires the use of a Docker Image file provided by Jupyter.org. That allowed us to set up a preconfigured container that already came with Apache Spark, including Pyspark, and Jupyter Notebooks. The initial setup required us to spend some time understanding the docker container environment, and forming that container as the starting point of our model. We selected a Pyspark-Notebook container.

Jupyter Docker Container

```
https://jupyter-docker-stacks.readthedocs.io/en/latest/using/selecting.html
```

Once our container was set up, we also added additional functionality to include a PostgreSQL server, and Pytorch with CUDA enabled. These are additional settings pulled in on creation of the container.

The Readme.md provides basic notes for running the application. To quickly get started, run:

```
docker compose up –build
```

The starting point of the application is found at:

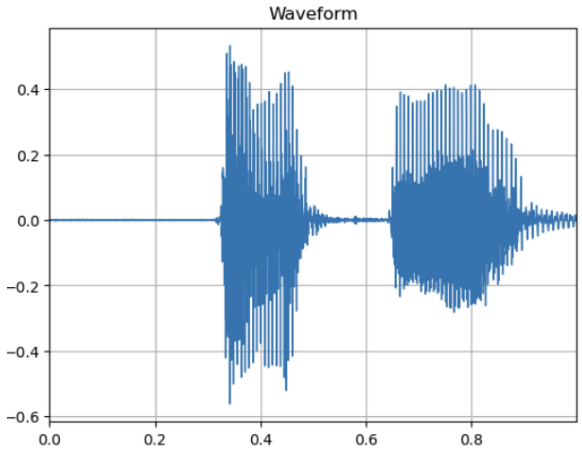
```
./speech_commands/CS4964-Final.ipynb
```

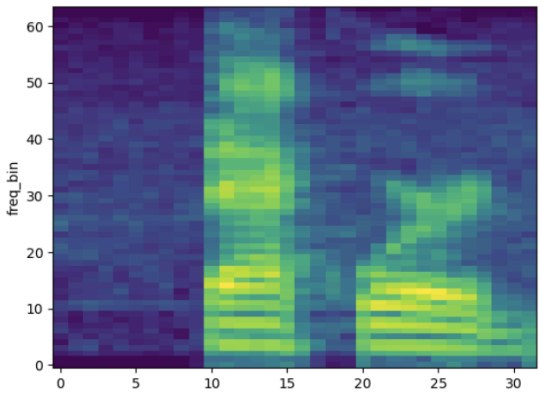
The notebook begins by calling into the HuggingFace datasets library to store the audio files on disk. Apache Arrow is used for data transfer and allows the audio files to be persisted, without pulling all of the data into memory at one time. We chose to leverage the capabilities here for the streaming of the audio files because of the efficiency, and we felt it would be redundant to reinvent the wheel. However, we continued to leverage the PostgreSQL and Apache Spark connections as part of the training loop of the Model.

Data Preparation

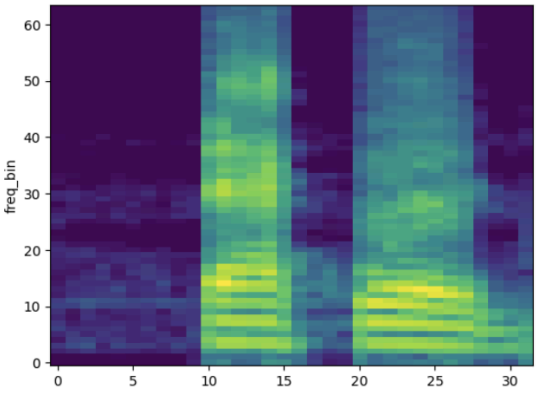
Static

After downloading the data - We began the data curation loop. There are two main things that we needed to do: We had to normalize the data, and change the array of the waveform into a 3 dimensional spectrogram to aid our model. Prior to this project, we did come in with little understanding of signal processing - but were enlightened to find that computer vision, and image processing techniques have paved the way for this field. We explored a few techniques for correcting some of the artifacts present in the audio files; such as silence, background noise, and static - and ended by using a technique known as 'Wiener Filter'. This technique corrects static by normalizing nearby samples from the input to soften the image. SciKit Learn, which is pulled in by the JupyterNotebook container, has this functionality available for use - and we leveraged it to build a spectrogram that was cleaned of noise.





Here we have the waveform, the noisy spectrogram and the cleaned spectrogram. The waveform has a lot of extra noise as visualized in the spectrogram on the left. Once we clean it we get the spectrogram on the bottom. This is much cleaner and the start and end of each syllable in the word is very clear. We hope that this difference will give us an accurate model for speech recognition. The Spectrograms are broken down by a Frequency Axis, a TimeAxis, and Amplitude. Brighter colors represent a higher amplitude occurring on the frequency bin at a given point in time.



Length

Not every sample contained 16000 data points. Few had less and even fewer had more. For these two cases we either added silence to the end of the sample or cut off the sample at its 16000th data point. From here we were able to change the waveform of the sound into the spectrogram form. Moving from the waveform that we were given, to

the spectrogram, we expected two main benefits: It would be easier for us to clear out the noise and the words themselves would be very distinct. Also the signal processing would be simple. With the added depth with the opacity in the spectrograms we could have one more feature we could pay attention to in our model.

Lastly, we applied an Amplitude to Decibel scale to logarithmically scale the inputs - this was to allow audio files that were quieter, or louder, to train utilizing the same scale as those that were speaking at an average amplitude.

Evaluation:

With our data cleaned we turned to a Convolutional Neural Network (CNN) for our inference model. CNNs are used widely for image recognition and speech recognition. With our use of spectrograms a CNN seemed like the clear choice to use. To train our CNN we used the built in functions from PyTorch, we didn't come from an extensive background in Neural Networks so we followed the Pytorch guide to implement it.

https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

During training of the model, we perform CrossValidation on the holdout set to test the performance. We output the accuracy, and the loss, at every 100 batches. At the end of each epoch, we perform cross validation and output that accuracy as well. We train a total of 3 models in our evaluation of the dataset. The first model we use as a baseline, taking in a spectrogram of the raw Wav file. The second model takes the denoised spectrogram input and performs training and evaluation.

We based our project on a similar basis as Pete Warden in his 2018 paper titled Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. Warden has a binary classifier between command and non command words. Our model runs speech recognition over 36 labels. The model outputs a vector of percentages of the likelihood that the given word is equivalent to one of the 36 labels. It chooses the label with the highest match or prediction and assigns that word to that label. This differs greatly from the binary classification that Warden does in his paper. This makes it much more difficult to get a similar accuracy.

After training the denoised model, we found the model inference performed poorer than the original, to our surprise. In an attempt to address this, and identify the cause - we utilized Apache Spark and PostgreSql by saving off the training data as we performed inference. We leveraged the File name, or the key, to group these data points together. We followed the work of DataCartography (Swayamdipta et al., 2020) to identify 3 types of training data: Hard-To-Learn, Easy-To-Learn, and Ambiguous data points. We were able to store those data

points to the local postgres server to continue to keep the memory footprint in our project low. At the end of evaluation, utilizing Pyspark, we were able to reconstruct the problems that occurred during training.

Figure 3: Data Cartography

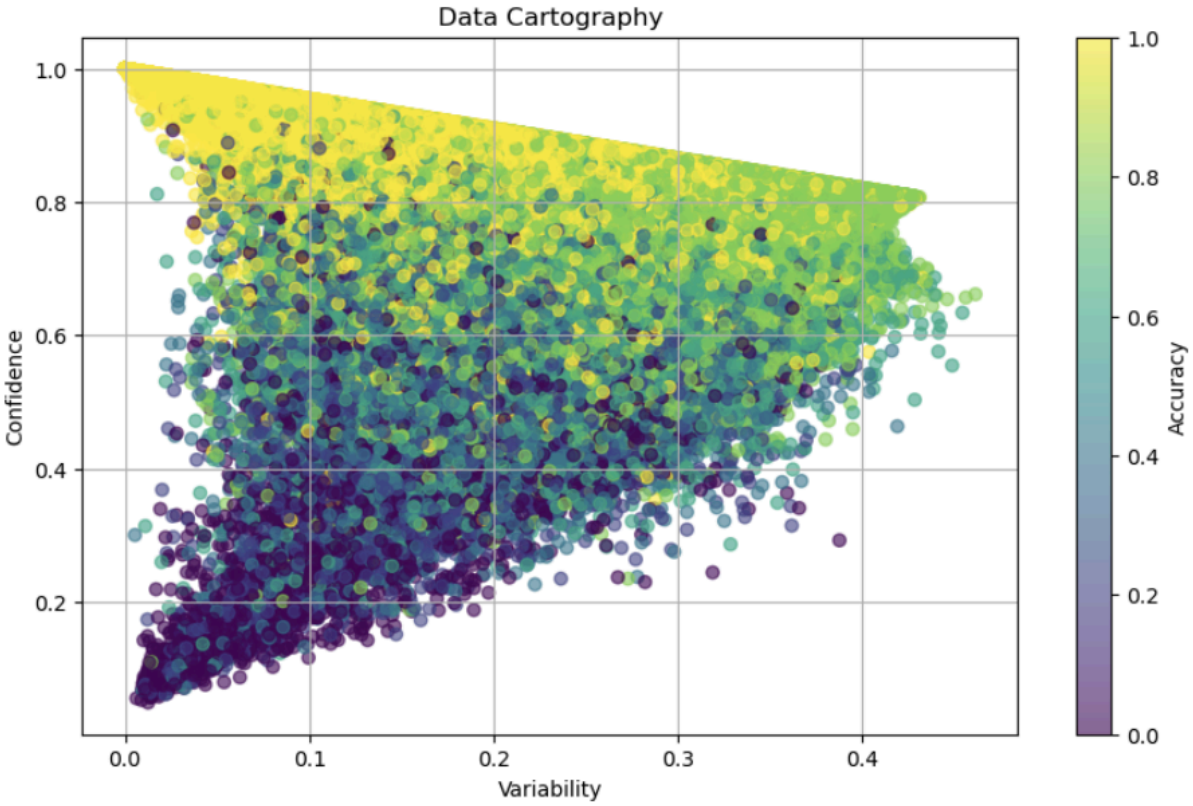
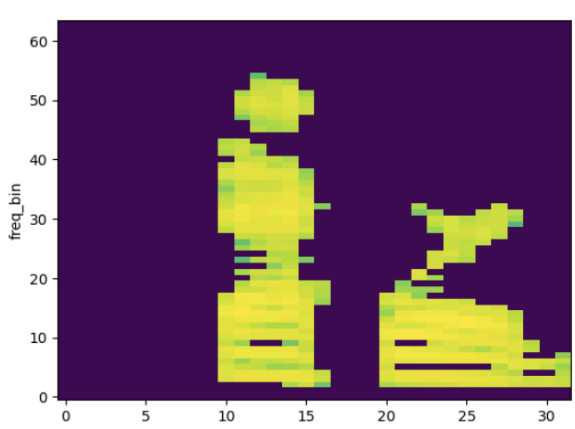


Figure 3 showcases the outcome of the training points that our model fails to evaluate on. We grouped the training Points based on the average Confidence, and Variability (Standard deviation of the Confidence across the training epochs). Our criteria for Hard-To-Learn and Ambiguous was data points that fell below 0.7 confidence and less than 50% accuracy across the epochs, or data points where the accuracy was less than 20% across all epochs. We created an additional data loader, with a sampler to prioritize these data points from batches at twice the rate of the easy samples. This yielded our weakest model unfortunately, at 71% validation accuracy. We wanted to attempt data cartography to showcase the use of Apache Spark, and leverage the Pyspark functionality.

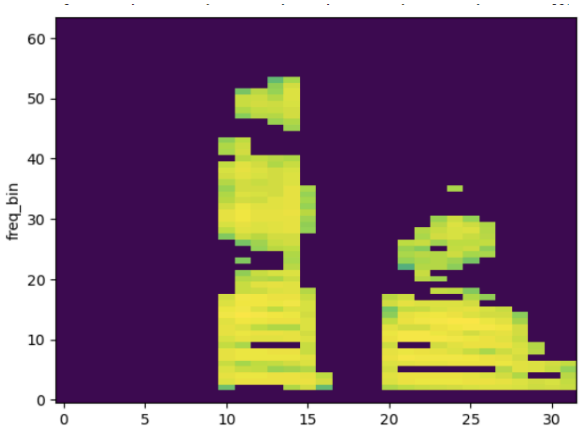
Conclusion:

The major point to learn from: simplify as much as possible without oversimplifying the input artifacts. We didn't find the results we wanted by cleaning the data and that could be because of the way spectrograms are used for. Spectrograms do differentiate between sounds very clearly such as how they are spoken and not what is spoken. However, words can sound similar and when spoken by the same person there may not be enough difference to reach the higher percentages of accuracy that we hoped for. Another cause of this is how we scaled the images with the logarithmic scaling. We can see that in this visualization from the non transformed spectrogram and the transformed spectrogram.

Non-Transformed



Transformed



The differences between the logarithmic scaling is caused by the denoising process. We attempted to fine tune this in an attempt to find a middle ground, but were unsuccessful. The frequency buckets are smoothed over and rounded out which could lose us information that the logarithmic scaling is able to showcase, and the model identifies. Another thing we believe is occurring is overfitting of the input data. As we were graphing the Accuracy scores against the Validation set at every epoch, we noticed between the 4th and 5th epoch that we began to perform more poorly. We had to limit the training time of our model so that this didn't happen and plummet the accuracy of the test data while doing great on the training data. The strengths of our project were the data processing elements, and the low memory footprint that we were able to achieve with Apache Arrow, Apache Spark, and PostgreSql. We were able to stream the data through the pipeline by utilizing the architecture that hugging face and pytorch provide, and showcase that we had an understanding of the hyper parameters that go into the training process that led to us being able to attempt to identify difficult, or mislabeled examples. Moving forward in this project it would be interesting to see how well we could distinguish different speakers based on the words they say and how they say them. Spectrograms seem to be well suited for that type model and distinction. We would also like to leverage the dataset to

add, or modify command words, and only create a probability distribution on certain words to see how well we can get the model to perform.

References

[(Swayamdipta et al., 2020)] Swabha Swayamdipta, Roy Schwartz, Nicholas Lourie, Yizhong Wang, Hannaneh Hajishirzi, Noah A. Smith, and Yejin Choi. 2020. Dataset Cartography: Mapping and Diagnosing Datasets with Training Dynamics. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 9275–9293, Online. Association for Computational Linguistics.

[(Warden et al. 2018)] Pete Warden. 2018. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. https://arxiv.org/pdf/1804.03209

[(Chen et al. 2015)] V. Panayotov, G. Chen, D. Povey and S. Khudanpur, "Librispeech: An ASR corpus based on public domain audio books," 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), South Brisbane, QLD, Australia, 2015, pp. 5206-5210, doi: 10.1109/ICASSP.2015.7178964.
keywords: {Resource description framework;Genomics;Bioinformatics;Blogs;Information services;Electronic publishing;Speech Recognition;Corpus;LibriVox},

[(Ardila et al. 2020)] Ardila, R., Branson, M., Davis, K., Henretty, M., Kohler, M., Meyer, J., Morais, R., Saunders, L., Tyers, F. M. and Weber, G. (2020) "Common Voice: A Massively-Multilingual Speech Corpus"