1. Introduction

We proposed creating an artificial intelligence agent that would be able to recognize vocal commands from an external source. The agent would be trained on a set of spoken words, a dict data type with an audio file and a label stating the word spoken. Once trained, the agent should be able to pick a word from a list of labels that is most similar to the sound given to the agent. We found interest in this project, because we did not work with audio in class and we believed it to be an important data type for AI to work with. Working with a more complicated data source, we wanted to have a simple problem, thus we created a simple classifier.

2. Related Work

Unlike many other projects the world of voice recognition or "speech-to-text" has been explored by many companies including Google's Cloud Speech-to-Text AI, Amazon's Amazon Transcribe, IBM's Watson, and Microsoft's Azure. Of course, these examples are commercial products where users can incorporate the AI into their own projects, so their research and development is presumably behind closed doors.

Overall, this confirms that our speech recognition is a topic and research area which helped us out when it came to understanding how to build an interpreter. This help came in the form of many forms including guides such as TensorFlow's *Simple audio recognition: Recognizing keywords* which has the basis for much of our work, Pydub for our audio processing from files, and much help from stack overflow for the bits between.

Since our dataset is connected with Tensorflow there are quite a number of studies that have used the same dataset with different methods and approaches such

as Peter Gao who creates some fascinating data visualizations (Gao, 2021). Or the Kaggle competition that offered a prize pool of $25,000 using a subsidiary of the dataset (Google Brain, 2017). And while we didn't use these resources directly in our project, it was nice to know that the data we were using is prevalent and applicable in many different uses.

3. Proposed Method

The goal of our research is to create an accurate classifier that can recognize human speech spoken into a microphone. The pipeline of this process includes selecting and loading the data, preprocessing the data, training the model, and then testing the results. We will also conduct tests with our own data to get an idea of the accuracy outside of the dataset to provide a better understanding of our model's reliability and accuracy when the data comes from an external source.

The data we chose was a collection of audio samples from a dataset hosted by TensorFlow. It contains 8.17 GiB of data when uncompressed and has splits fit for training and validating the data, as well as testing the data. We chose this data due to its size and sample variance in the hopes it would influence the final results of our model to be better equipped for voice variance. The data is stored in a raw audio format that represents a waveform showing the frequencies of that waveform over a set period dictated by the sample rate.

The act of processing the data had a variety of steps from loading our data, shuffling that set, creating batches for our model to read, and shaping it so that the model could read it without running into any issues. In doing this, we found it important to filter our

data to remove any samples that were not at the same sample rate as the majority of the data. We did this because we wanted our model to focus on learning the patterns of speech instead of the metadata of the speech such as how it was recorded or the volume associated with it, therefor the data was shaped to only include audio files with a sample rate of 16,000 or 8000Hz.

One other process we accomplished was training our model both with values with the label unknown, and without values labeled unknown. We found that this was due to the nature of the data not being incohesive speech or silence but rather commands that were not labeled in the dataset. Such examples include words like “four” or “blue” that are physical words that could be labeled but were instead lumped into the unknown label. This caused problems for our model’s predictions because it would mark all of the data as unknown. Therefore, we recorded our model both with and without the unknown label.

Our data was then shaped to become a spectrogram rather than a waveform to assist in the pattern recognition we wanted our model to present. Our understanding was to treat this problem similar to the classic MNIST problem that we have worked with in the past. The line of reasoning is that if we could convert our sound into a picture, then the same methods of understanding would apply to both MNIST as to our speech commands. Therefore, the data values were shifted to form steps of 128 that the model could read instead of a constant input of values.

As mentioned above, our model used 2D convolution to recognize patterns of a spectrogram to predict future inputs presented to it through a file. Our method of

implementing the model was done through TensorFlow to help us get better results. Therefore, much of the work was put onto the TensorFlow program and we did not have control over. Our actions were to resample the data, run it through 2 Convolution2d Keras layers with MaxPooling2D and dropout, followed by 2 flattening layers with the first having dropout and the second resulting in a predicted label. All of our layers were ran through the rectifier application as we were already straining the memory of our online environment. After that we would fit that model to our data with validation to analyze the results.

After running our model on our test data we would consider our results through looking at the accuracy score and confusion matrix to better understand the words and patterns our model had considered. We will also run tests through uploading files of our voices saying different commands to see if the bot could properly understand us as we spoke, as our voices were not present at all in the dataset.

4. Experiments

We had to go through many experimental methods. We had to find what was the best way to preprocess our data for our classifiers to work, and we had to find the best classifier for that data. The data always came in as a .wav file. We began by converting the .wav files into numpy arrays. With these arrays, we classified the data via a multi-layered perceptron (MLP) classifier. This produced an abysmal accuracy and was quickly scrapped. Our next attempt converted the audio to a TensorFlow dataset, then using TensorFlow methods, we could create a spectrogram from the data. The spectrogram was thus more simple for classifiers to interpret. We could put it through an MLP classifier, which was more accurate this time around, but not as accurate as the

classifier offered through TensorFlow: the convolutional neural network (CNN) classifier. The CNN classifier ended up giving us an accuracy above 70%. Thus, our model became that using TensorFlow's spectrograms and CNN classifier.

With our model built came the decision of how to further preprocess the data. With the dataset we were given, the labels: 'down', 'go', 'left', 'no', 'off', 'on', 'right', 'stop', 'up', 'yes', 'silence', and 'unknown', were given for a data set of audio files of people saying words beyond these labels, those words were labeled as 'unknown'. We then had to make the decision whether or not to remove all files labeled as 'unknown'. If we removed the label, we would lose the ability for our agent to ignore an unintelligible audio file, however it would remove a lot of noise from our training data set. When we included 'unknown', as shown in our confusion matrix, our model was very quick to label words as 'unknown', even those that the model without 'unknown' could detect. Our accuracy was much higher on the model when we removed the 'unknown' values, so if we were to continue further, we would continue with that model.

Further experimentation was required to create audio that the model would be able to work with. We tried using a provided TensorFlow method to record and process audio from a microphone, however computers have different hardware and abilities, thus making this method impractical. We then found that we could record audio in Audacity, which also gave us more options to edit the file in order to fit it in our model, and then put that .wav file into our program which would process the data and output a prediction as to what word was said.

5. Results and Discussion

When we work with our completed model, by importing the audio files into the program, we are able to label the word correctly fairly often. If one considers the model trained without the 'unknown' labels as the final model, it was able to categorize 6/6 audio files that we tested it with, as shown during our presentation. Although tested against the given dataset we were only able to produce an accuracy of around 60% (we were unable to generate a seed for what data we grab, so there is some variation between sessions). It can be assumed that if the examples provided were less clear when spoken, we would not have 100% accuracy. Using a live microphone is something we would look into, but we did not have the time to completely tune up the process.

6. Conclusion

This project was a fascinating look into a practical use of Alternate Intelligence. The results of our model were pretty successful, but still has room for improvement and modifications to the listed commands. Although, realistically speaking, the research we've done is nothing groundbreaking when compared to the work other voice recognition programs have accomplished. Instead, this was a great learning experience into understanding how computers represent audio and how programmers can modify and train that audio.

7. Contributions

The bulk of our final code was produced by Daniel. Richmond and Caleb worked on early prototypes that assisted with the beginning of the project, however that work would end up not being included in the final product. The soft stuff was handled by

Caleb, this includes the presentation and some of this paper: sections 1, 4, 5, and 7.

The rest of the paper was finished by Daniel.

References

Gao, P. (2021, April 25). *Deep Learning For Audio With The Speech Commands Dataset*. Retrieved from Medium: https://towardsdatascience.com/deep-learning-for-audio-with-the-speech-commands-dataset-79f7023de903

Google Brain. (2017). *TensorFlow Speech Recognition Challenge*. Retrieved from Kaggle: https://www.kaggle.com/competitions/tensorflow-speech-recognition-challenge/data

pydub. (n.d.). Retrieved from github: https://github.com/jiaaro/pydub

Simple audio recognition: Recognizing keywords. (n.d.). Retrieved from TensorFlow: https://www.tensorflow.org/tutorials/audio/simple_audio#convert_waveforms_to_spectrograms

TensorFlow Developers. (n.d.). TensorFlow (v2.8.2). *Zenodo*. doi:https://doi.org/10.5281/zenodo.6574269

Warden, P. (2018). Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *ArXiv*. doi:https://doi.org/10.48550/arXiv.1804.03209