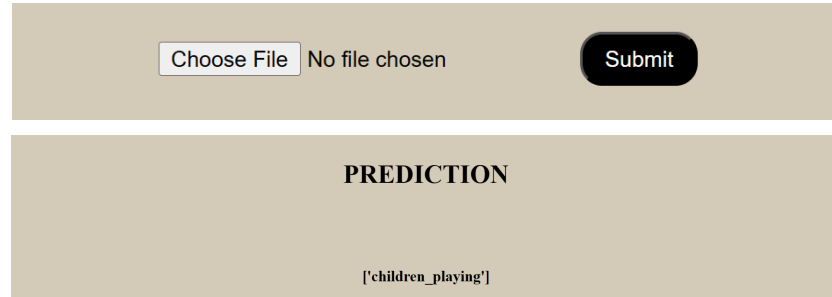June Shao
Claire Yuan

Audio Classification: Writeup

We created an IoT system where a user uploads an audio file from their laptop to a web app, the file is sent to the server, and the file is then processed with machine learning to predict what type of noise is in the file. Lastly, the server displays the prediction on the webpage.
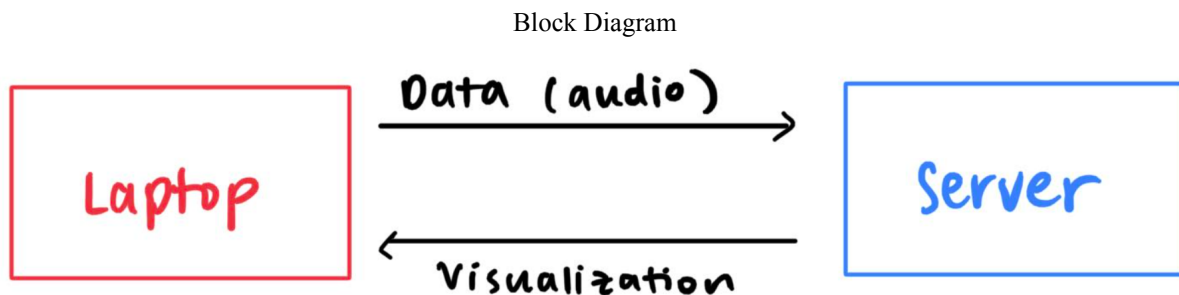
User interface

Block Diagram

We created our server on Render. The IoT system consists of two physical nodes: the user's laptop and cloud server. The user node is simply the frontend of the web app. The server on Render is where the website is hosted and the data processing and machine learning takes place.

To build our web app on our server, we used Python and Flask, employing the HTTP protocol. When a user navigates to the website, a GET request is sent to the server, and in response, the server displays the HTML template for uploading a file. When a user submits a file, a POST request is sent; the server then gets the audio file from the request and uses a previously-trained machine learning model to predict what type of sound it is.

Key Processing Techniques

Before building our web app, we first created a machine learning model using TensorFlow and Librosa (a library for audio analysis), based off of code from https://www.section.io/engineering-education/machine-learning-for-audio-classification/. This code was in pieces, had errors, and also was not made for a full python file. Therefore, we needed to understand the bits and pieces of the code in order to rearrange and make edits. We downloaded a dataset of around 8 thousand labeled audio files to split them into a train set and test set for our model. The network consists of a linear stack of 4 layers; the last layer is a softmax layer to convert the vector into a distribution of probability for multi-class classification. Lastly, we trained and tested the model. The model has an accuracy of 70%, which we saved so that it could be simply loaded and used for future predictions in our web app.

Our code relied on the Mel-Frequency Cepstral Coefficients (MFCC) algorithm, a technique for extracting the frequency information of the audio signal across time. To process the user's audio file, we used Librosa. Librosa first loads the audio file into a numpy array and finds the sampling rate. Then, a different librosa function converts this loaded file using MFCC. This MFCC output is then inputted into the pre-trained machine learning program. It then outputs what class of noise it believes it is.

For visualization, we just used Flask to update the HTML text and display the prediction.

Design Choices

We decided to use Flask because it provides a straightforward way for the server to receive a file from the user, process it with existing Python libraries, and manipulate the HTML.

Reflection

One limitation was that librosa needed a file path. However, when the user uploaded a file there was no file path. This was resolved by downloading the user file into an "uploads" folder, and having librosa read from there. However, this failed because Render seemed to not allow for user uploads. Therefore, we had to go back and change the code to extract the file name and file in a way that librosa could use as a parameter. Later on, the website would always error out. This was solved by increasing the timeout time with gunicorn.

Another main issue is that the outputs of the ML model may not be accurate depending on how the audio was recorded. Frequencies can become slightly distorted in some cases, or there may be undesired noise (e.g., wind blowing into the microphone). The dataset we used had some flaws, such as bird sounds in an air conditioner audio sample. Discrepancies in the dataset audio and user audio would cause inaccurate results. If we trained our model with a more representative dataset, the accuracy of the results for general users would be better. Also, we can try adding more or less layers, changing the layers, and playing around with the number of epochs.