Keyboard Decipherment

Daniel Ryaboshapka, Taylor Korte, Harris Hardiman-Mostow

Table of Contents:

1.0 Introduction:

Given the prevalence of audio-reliant virtual assistants such as Siri or Alexa, a machine's ability to identify and interpret audio signals is uniquely important in the modern world. Motivated in part by this desire for convenience that has inundated the marketplace, our research explored combining machine learning with acoustics analysis to determine what a person is typing in real-time, using only the user's on-board computer microphone. While we initially wanted to experiment with a wider array of microphones, due to the COVID-19 pandemic and transition to distance learning, we had to narrow the premise of our research question to only using the built-in computer microphone, specifically those on MacBook laptops. This research project was further motivated by our interest in investigating audio and waveform analysis and applying machine learning principles to make accurate predictions, as well as the possible cybersecurity implications that may result from accurate keyboard press recovery. The results of the project may lead to insights in predictive software in audio analysis.

Existing literature and research done at the University of California, Berkeley suggests that it is possible to train a Machine Learning model to recover keystrokes with high accuracy and display the computer's prediction in real time (Zhuang, Zhou, & Tygar, 2005). While Zhuang, Zhou, and Tygar were motivated by password-security concerns arising from these results, we are interested in an algorithms ability to recognize audio samples from sparse information - namely, audio collected from one on-board microphone - and how that can be used in the context of audio-reliant services.

The fundamental theory of importance to us during this project was the Fourier Transform, which allowed us to decompose audio signals into their constituent frequencies and better enabled us to analyze frequently noisy data. In particular, we made use of MATLAB's Fast Fourier Transform command, `fft`, along with Python's Scipy, `stft`, or Short Time Fourier Transform command. We used TensorFlow/Keras as our training backend as we trained a Neural Network that eventually determined the key presses from the audio data. We filtered the audio signals of all keypresses into multiple streams for the network to classify. Neural Networks are omnipresent in much of Machine Learning and are well-studied. To give sufficient background would distract from the report. At a high level, they are constructed to "learn" from their errors in prediction and classification (Mendels, 2019), and hence are a good structure for our purposes. Our hypothesis is: Given a large sample of training data, our model can, within a tolerance for error, accurately locate the sound of key presses in three-dimensional space and thereby determine what key is being pressed at that moment.

2.0 Methods:

Our setup for collecting data required only a laptop with an on-computer microphone. In order to simplify the post-processing of the audio, we limited the laptop type to be any Apple Macbook Pro later than 2015. This ensured that the microphone placement and quality is similar for each set of data. For one of our datasets, we utilized this microphone to record a person typing a set of 15 random words from a dictionary of 900,000 words (in our directories this file was called words_alpha.txt). In addition, while the user typed, the Python code collected a list of each key typed and its corresponding time. We collected data from three different people and obtained five trials from each. This gave us 2700 unique waveforms to analyze. Using these waveforms, we could identify the letters that corresponded to certain waveform shapes and patterns, especially after the data was put through a Fourier Transform. In order to analyze this data, we used Fast Fourier Transform, Short-Time Fourier Transform, and Mel-frequency cepstral coefficients (Mendels, 2019). By using these analysis techniques, we could determine the frequency and duration of a keystroke, which constituted the backbone of our machine learning algorithm.

In order to recover what a person typed, we trained a Neural Network to identify letters, backspace, and the spacebar using a set of training data, which consisted of a random sample of key presses and corresponding audio. The first training dataset consisted of only the backspace and the spacebar keys. Our second dataset recorded a person typing the first row of letters randomly. For the third dataset we recorded each

letter being pressed on the keyboard. The final dataset consisted of random words. This way, we could verify our accuracy as the project went along. The neural networks were constructed using TensorFlow and Keras. Our initial model used the raw audio data as an input and utilized Least Short Term Memory nodes to process the amplitudes and outputs to an On-Off node. The model for the entire keyboard also fed in raw audio as the input, but it immediately converted the data into a Log Mel Spectrogram (Figure 1). In order to better process this data, we utilized standard image processing techniques (instead of just using raw audio data) using convolutional layers and outputted to one of 28 classes (26 letters and space and backspace keys). For each of our models we used additional datasets to validate the model.

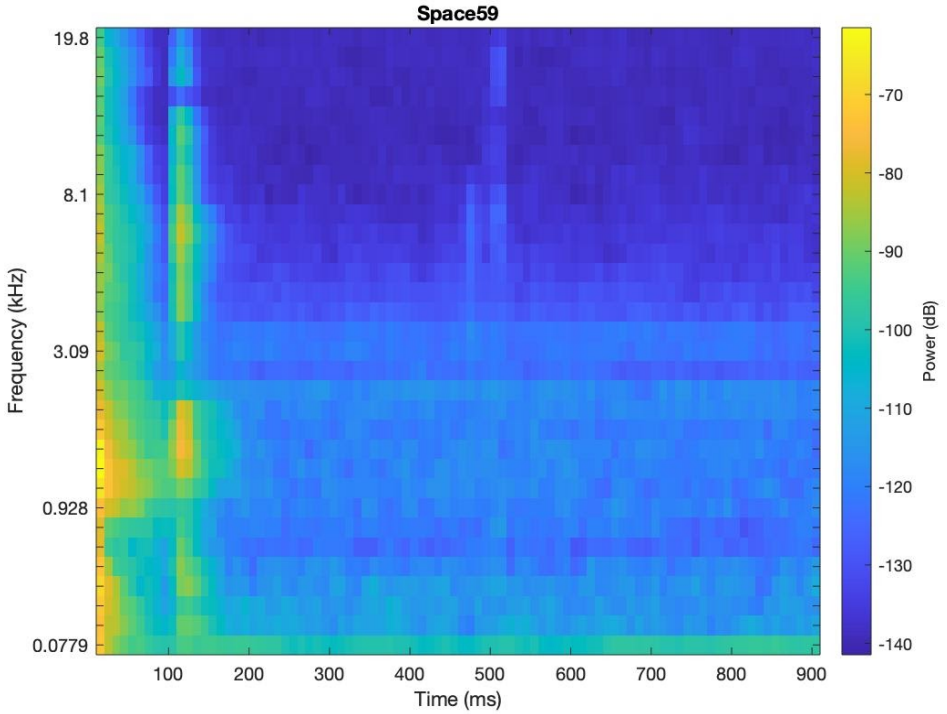


Figure 1: Log Mel Spectrogram for Spacebar Key Press (note: two yellow bars indicate initial press and return of the key)

3.0 Results:

	Backspace and Space Key Model	Top Row Keys Model	Entire Keyboard Model
Fit Accuracy	65-85%	99%	99%
Number of Fit Data Points	100	256	2800
Validation Accuracy	55%	35%	60%
Number of Validation Data Points	20	100	700

Table 1: Accuracies and number of data points for each model

The figure above shows that the backspace and space key model has a lower accuracy as a result of the different types of models used. The model for the backspace and space keys only compared the audio data whereas the model for the top row keys and the entire keyboard compared spectrogram images. Combined with the increase in data points, these two factors resulted in a higher validation accuracy for the latter two categories. For the entire keyboard model, we used data from two people using two different laptops. This resulted in a lower accuracy than expected because of different background noises, microphone strengths and typing techniques. Despite this, the model had the best accuracy out of the three categories, due to the large sample size of data points. The 2800 data points also lends itself to the statistical significance of our results.

4.0 Impact and Conclusions:

While the COVID-19 pandemic forced us to make some adjustments as our project progressed, we were still able to generate meaningful results and document our scientific process thoroughly. This project challenged us to use several interesting acoustic analysis techniques, such as the Fast Fourier Transform and Mel-Spectrogram, as well as to create a complex machine learning algorithm.

The potential impact of this project spans many fields, including audio recognition and collection, neural networks, and the integration of the two. More broadly, machine learning, which neural networks are a subfield of, continues to be one of the most active fields of research in engineering, mathematics, and computer science. As we alluded to earlier, ever-present virtual assistants such as Siri or Alexa makes these topics particularly relevant outside of academia as well. These virtual assistants rely on both recognition and predictive capabilities; while our project focused on recognition, the results and methods could be extended to prediction as well. In addition, this project makes it clear that computer manufacturers need to take security precautions when constructing their devices; if a third party were to gain access to a laptop's microphone, passwords could easily be stolen using techniques outlined in this paper. Moreover, while our project focused on audio signals from keyboards, certainly the results of this project is further proof of the viability of researching speech recognition, another application combining machine learning techniques with audio

signals. While more complex, this would have a wide-ranging impact on people with speech or auditory disabilities.

5.0 Logbook Appendix:

3/31/20:

In our first lab meeting since before Spring Break, we gathered as a whole lab section to go over the new lab formats and discuss how each group would adjust to the new COVID-19 reality of trying to execute a final project. We broke into our project groups via our own Zoom channel. We discussed our revised project ideas with Professor Huang, who liked the direction of our project, while noting that he wanted to make sure we emphasized real-time continuous data collection and analysis and other such topics covered in class. After checking in with TAs Martin and Molly, Dan showed us the Python code he had created so far. While Dan continued his work on the code, Taylor and Harris worked on gathering a suitable collection of words and sentences that would serve as the prompt for our test subjects to type and saved them to Plain Text files so they could be integrated into Dan's program. We also created a shared document where we could write down and keep track of a to-do list, as well as log our activities during the lab block for future reference.

Taylor and Harris also installed the necessary software onto our computer in order to run the code (various Python-related extensions), which turned out to be more of a hurdle than we expected, so we might have to rethink how we are going to deliver the test to a wide range of subjects now that everything needs to be remote.

Finally, Dan, Taylor, and Harris were each able to run 10 trials of "audio_test1.py". Each trial generates an audio file and a file with the words the user was prompted with compared to the actual words typed. Next lab, we hope to review these files and go over the software necessary to analyze said files; from there, we can continue to implement the rest of our project.

4/7/2020:

In this session we worked on analyzing data from our initial testing. We translated the .wav files into plots of amplitude versus time to begin analyzing. Taylor and Harris worked on a MATLAB code to detect the peaks of the code within a certain threshold, which we hypothesize are typically when the test subject presses the spacebar key. By doing this, we have a good starting point to begin breaking down the rest of the keyboard by identifying the beginning and ends of the words, signified by the pressing of the spacebar.

We refined the peak code several times, and the output is shown below.

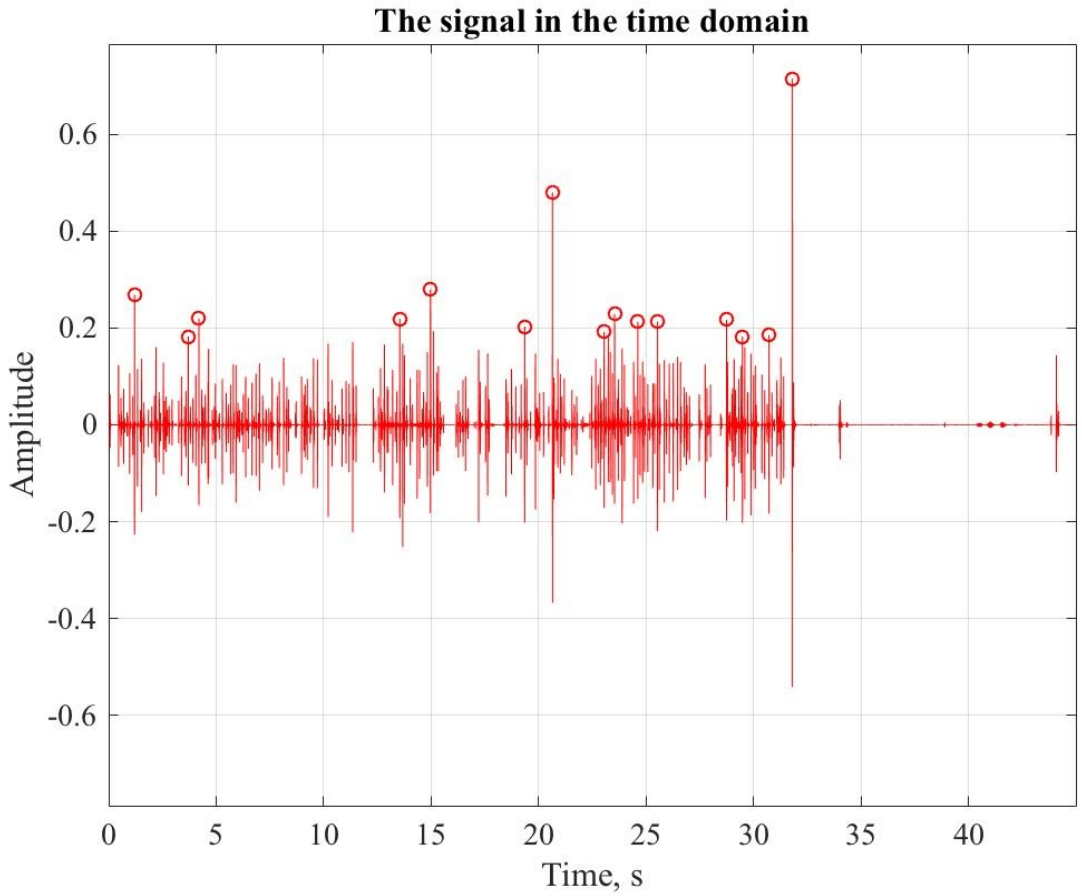


Figure 1: A amplitude plot of the 15 maximum peaks in the dataset

This is a good start, but it appears to miss a couple space bars in the 5-15 second range and too many in the 20-30 second range. We want there to be exactly 15 peaks – 14 for each spacebar in between 15 words, and then a return key when the user finishes.

This issue with this rendition of our project and the code is we cannot connect the dots between the above audio file and the keys or letters being pressed, or to validate if our data analysis is correct. So, we realized we needed to collect more total data in each test so that we had more of a reference of what keys are being pressed at what exact time, which will allow us to refine the actual ML model more efficiently. Dan is currently developing a python code that implemented this and also displayed a better interface (Figure 2) so that we can collect data from a wide range of people without a lot of confusion. We will try this out in between this lab and the next one and do some post processing on the new data to see if it is enough information to continue with the ML model.

Next week, in addition to testing out the new python code for audio data collection, we want to possibly investigate Linear Discriminant Analysis as a possible method for classifying key presses, which Professor Huang pointed us to. We also might try to implement a control test where the user presses every key of interest to us before running the test.

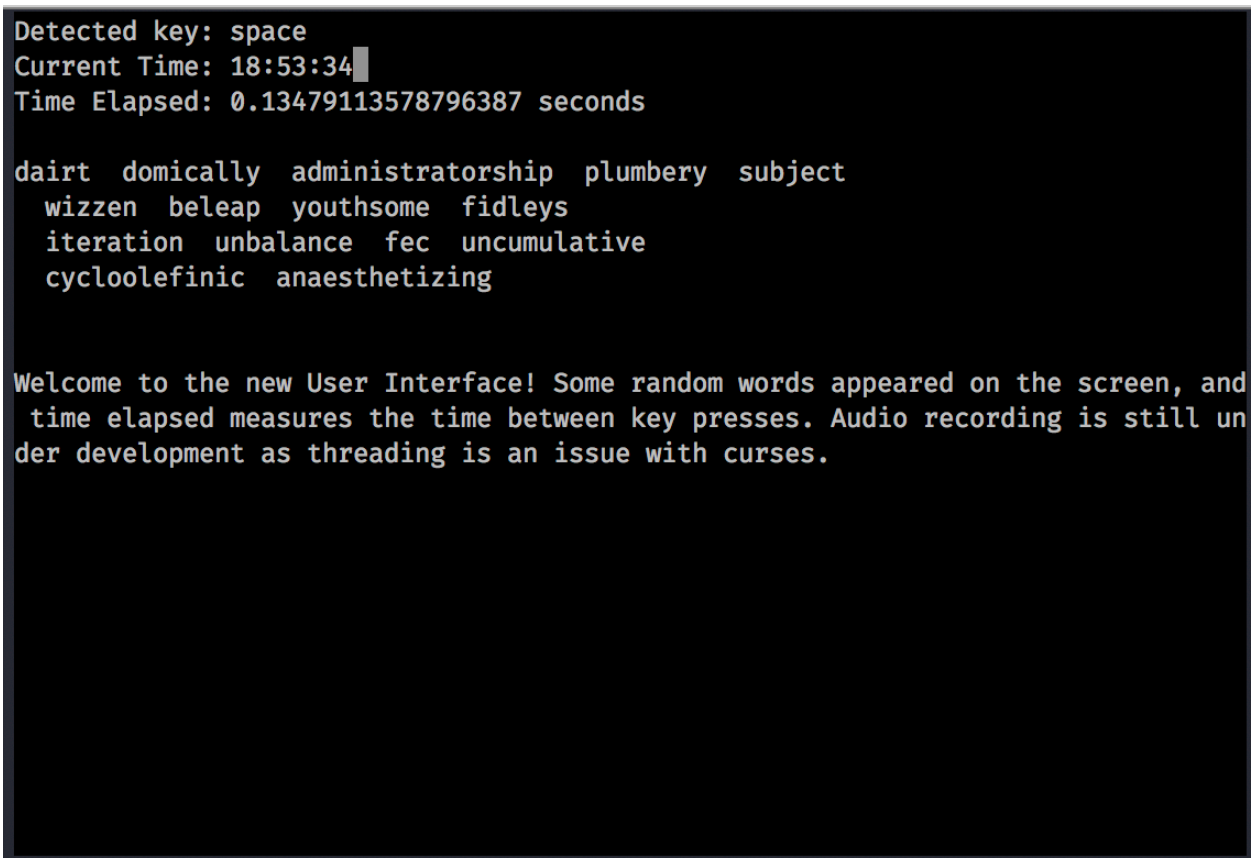


Figure 2: User interface for Python typing test

4/14/20:

Before this meeting Dan created a new code that measures the time between each keystroke when typing random words in addition to the .wav file (created in python). It also recognizes the key being pressed. This will allow us to connect the dots between the audio recording produced after each trial and the actual keys that correspond with the audio, which solves the problem we identified in the previous lab session. Each of us ran the program 5 times in order to get a good set of data to begin analyzing and post processing. Below is a sample output of the audio file overlayed with the instances of keystrokes (Figure 1).

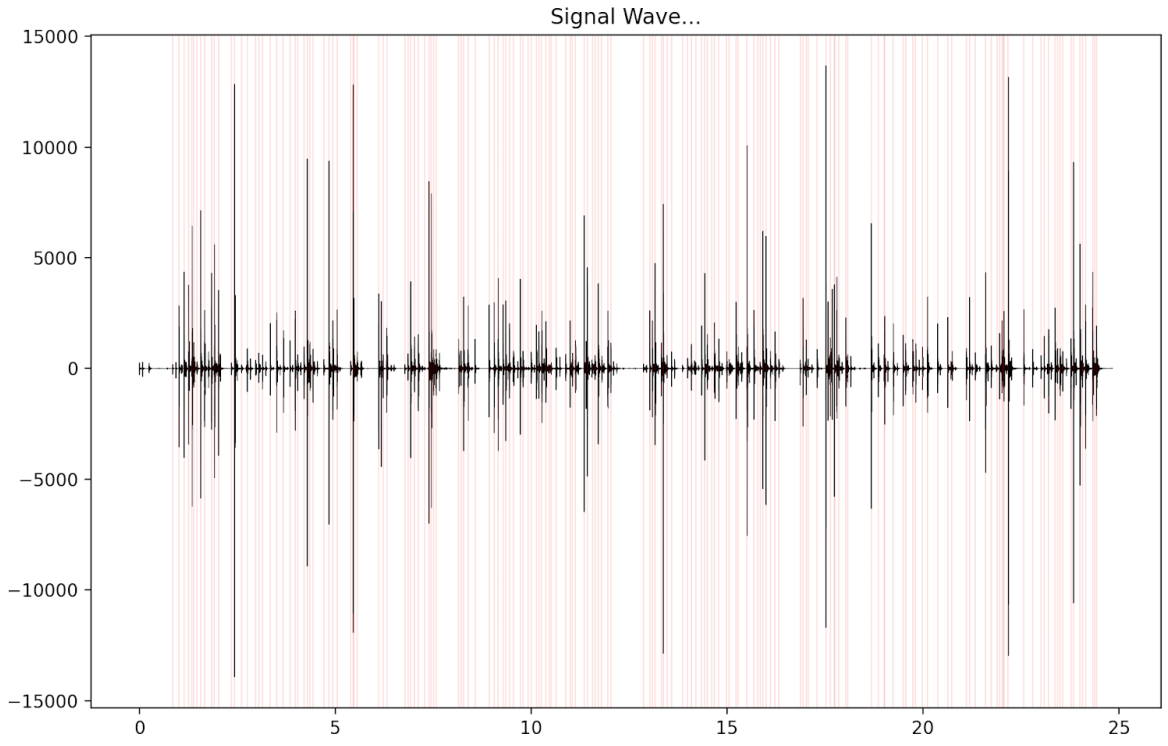


Figure 1: Amplitude plot of keystrokes with time of each keypress in red

Professor Huang recommended trying to do some sort of Fast Fourier Transform on the audio data that might make it easier to work with in post-processing. We were having difficulty doing this with high precision because the data files are about two million data points long and

so any sort of moving loop would take forever to run, so we had to make the sampling fairly coarse to make the FFT practical to use. We eventually got it working, however we're not exactly sure if or how this will be useful. We might try to overlay it on the plot Dan created with the indicators for letters being pressed. It might make the analysis and identification of letters via audio (or Fourier transform of audio) more efficient or accurate. An example of the output plot of this code is shown in Figure 2.

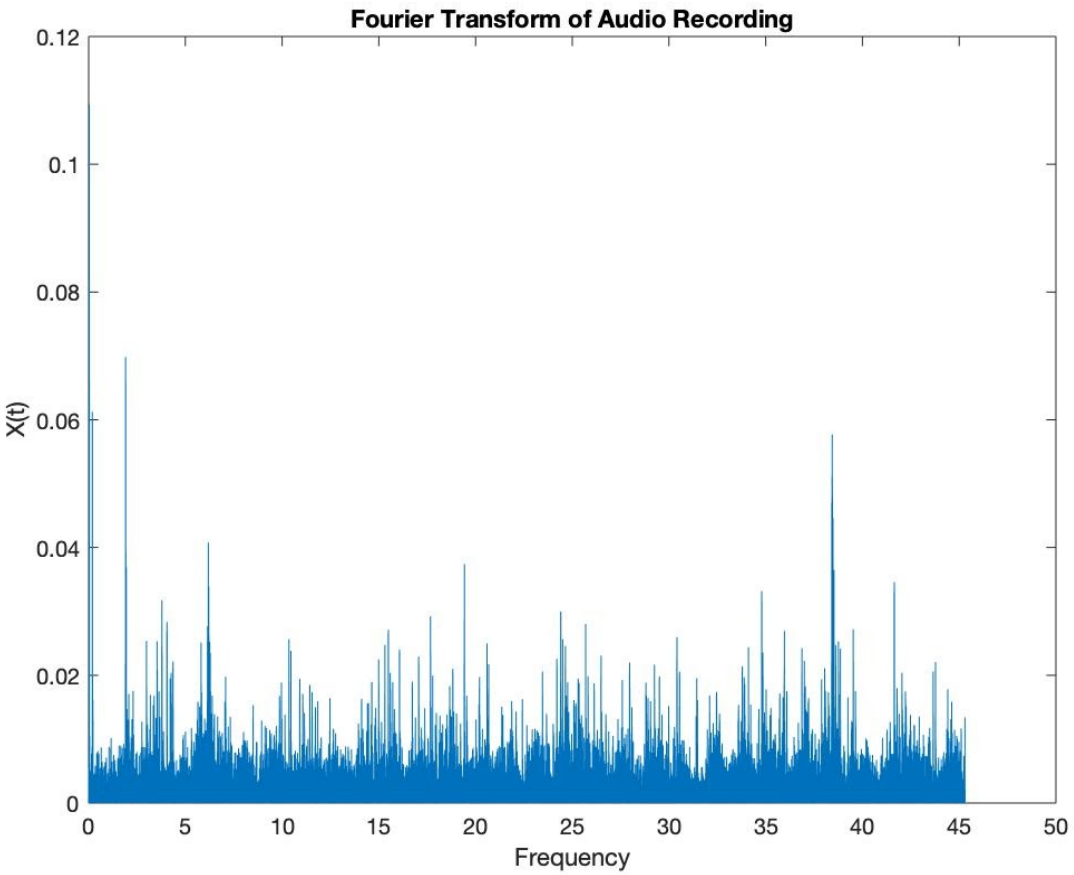


Figure 2: Fourier transform of the keystroke recording

Next, Dan separated the sound file into each individual keystroke (an example of the letter d is shown in Figure 3). Using these files, he created subplots (Figure 4) of each keystroke so that we can analyze the differences between keys. For example, we noticed that the h key has a relatively similar shape each time it is pressed.

We decided that given the time and resource constraints we will not be able to gather test data from people outside of our lab group. This is disappointing but we think it will allow us to focus on analyzing the data we already have.

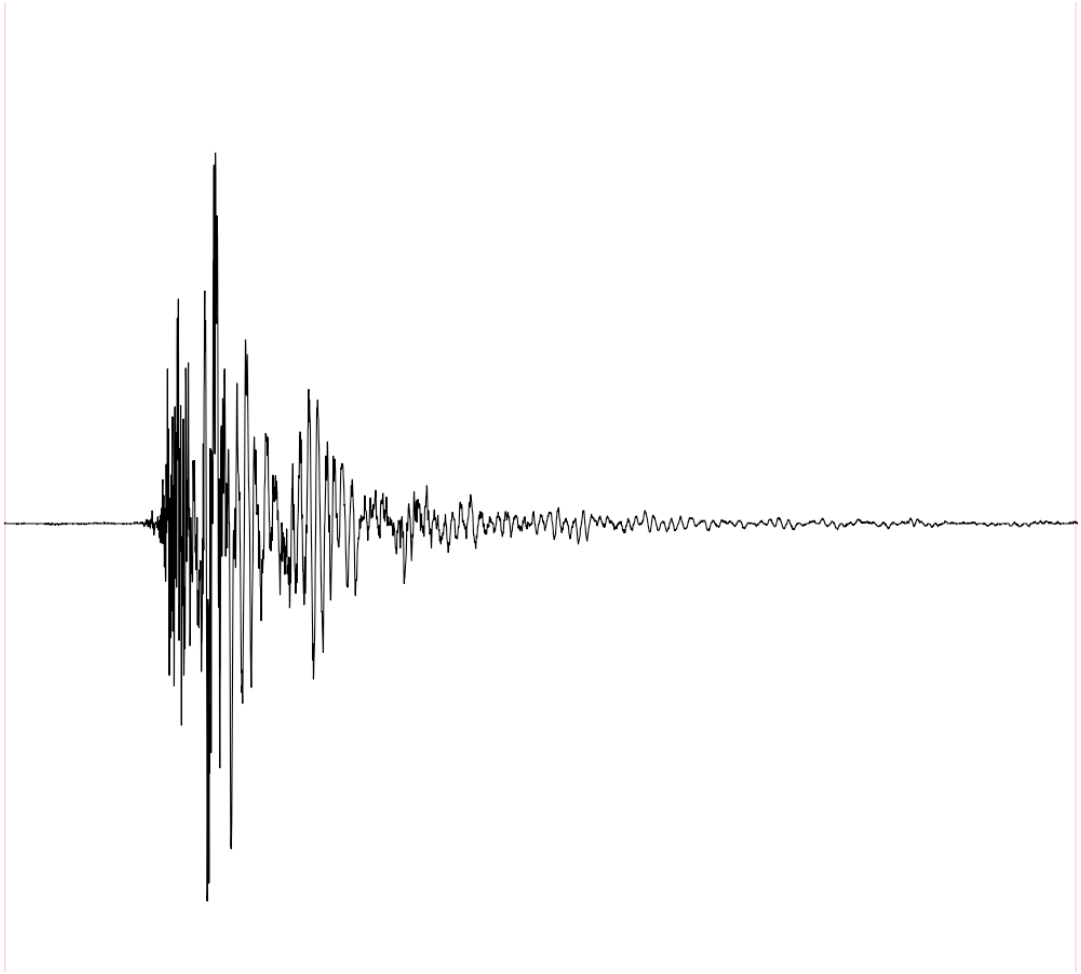


Figure 3: An isolated sound wave of the letter d being pressed

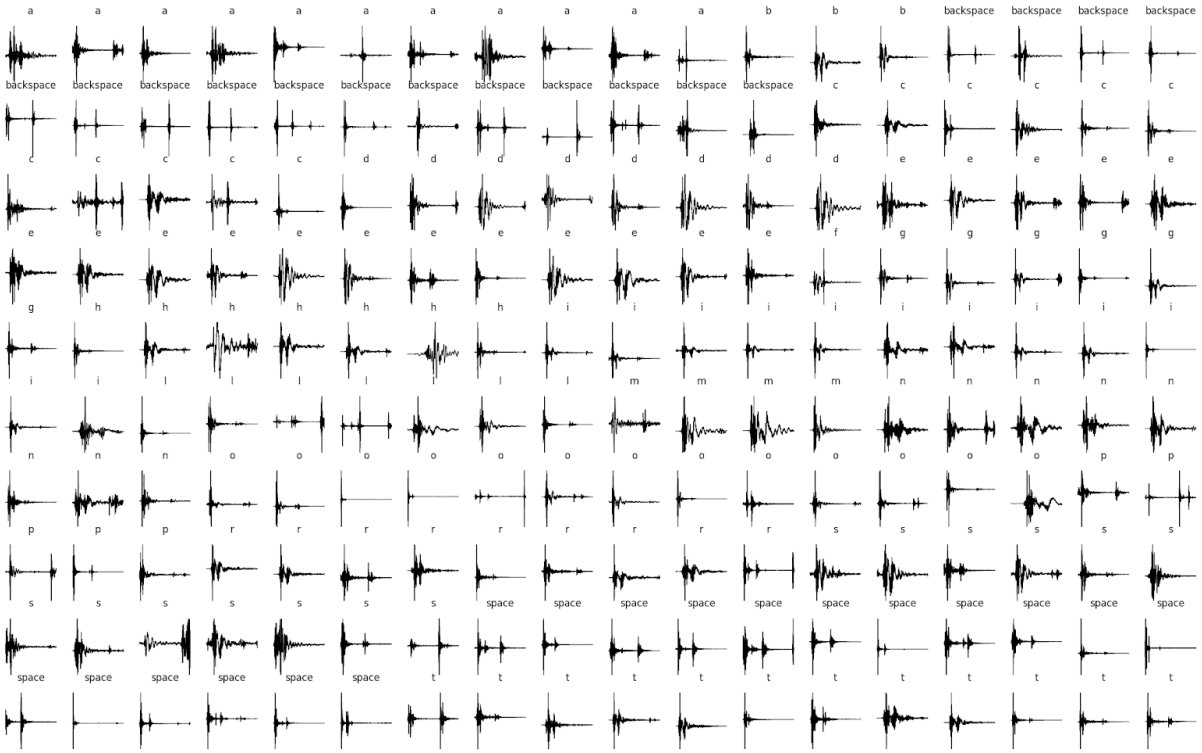


Figure 4: Subplots of the amplitude for each key pressed in the recording

4/21/20:

For our last lab meeting we decided to continue analyzing our data in MATLAB and writing our lab report.

Taylor started that lab by looking at the fast fourier transform (using code from the last lab) of the "letter snippets" that Dan created during the last meeting. At a glance it didn't look like there was much of a pattern or discernable difference between different letters, which is a little worrying/frustrating, but we are going to keep collecting data and hope the model will identify patterns when we train it.

Harris updated the final report so that it followed Huang's specifications. For example, using past tense for the most part and present tense when relevant, an introduction that focused on motivation and existing state of science rather than "background", and avoiding passive voice to increase clarity.

Dan worked on the Tensorflow model so that we can start training the algorithm. We want to start this ASAP but are unsure about how we want to continue given that the Fast Fourier Transform isn't producing the patterns we were hoping for.

We discussed implementing a filter on the data in MATLAB, so the FFT might produce something more desirable. In addition, Dan possibly suspects something is wrong with the alignment of audio and with letters and his goal is to begin to use Tensorflow to group our data. Dan will try to run the model with the raw audio data and see what algorithm produces. We also realized that we may not need the fast Fourier transform to run our model.

5/3/20:

We have stopped meeting as a lab group, but have continued to work individually.

Dan:

Created a model of space and backspace (code is below) using the following steps:

1. Run **typer.py** to output two files: an audio file (.wav) and data about the time keys were pressed during recording (.csv)
    a. Example Run: typer.py -w spacebackspace.wav
        i. Outputs spacebackspace.wav and cornutin.csv (a random word generated from the random words given within typer)
2. Run **plotwav.py** after modifying the csv and sound file targets within the script to generate a plot of the overall sound file with vertical asymptotes over every part where the typer suspected a key press. Outputs all of the cropped sound files (based on these vertical asymptotes) to a directory specified within the script.
3. Modify **train2.py** (initially for spacebackspace, changed to **melspectrotrain.py** for toprow and full keyboard analysis) and **makemeta.py** (code to create the metadata csv file for the data pipeline when training the model) to use the directory full of sound files created from step 2. Modify parameters (steps, learning rates, epochs, batch size) and begin training based on the amount of data created from step 1. Data pipeline adapted from Schewrtfeger.

<u>Spacebackspace Model:</u>

```
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.BatchNormalization(momentum=0.98,input_s
hape=(44100, 1)))
    #
model.add(tf.keras.layers.Bidirectional(tf.compat.v1.keras.layers.CuDN
NGRU(128, return_sequences = True)))
    model.add(tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(16,
return_sequences=True)))
    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(1,activation='sigmoid'))
    opt = tf.keras.optimizers.Nadam(learning_rate=0.001, beta_1=0.9,
beta_2=0.999, epsilon=1e-07, name='Nadam')
    model.compile(optimizer=opt,loss="categorical_crossentropy",
metrics=['accuracy'])
```

After initially trying to run the model on our laptops, we predicted that the model would take 37 hours to complete. We realized that our laptops do not have enough processing power

and the data pipeline (the way of sending audio files to tensorflow models) is very disk intensive (limited by RAM). As a result, we utilized Google Cloud Platform in order to create the model. This platform allows us to use the GPU's virtual memory to speed up the process. For this model the accuracy was 85% which we thought was relatively high for a dataset containing two keys.

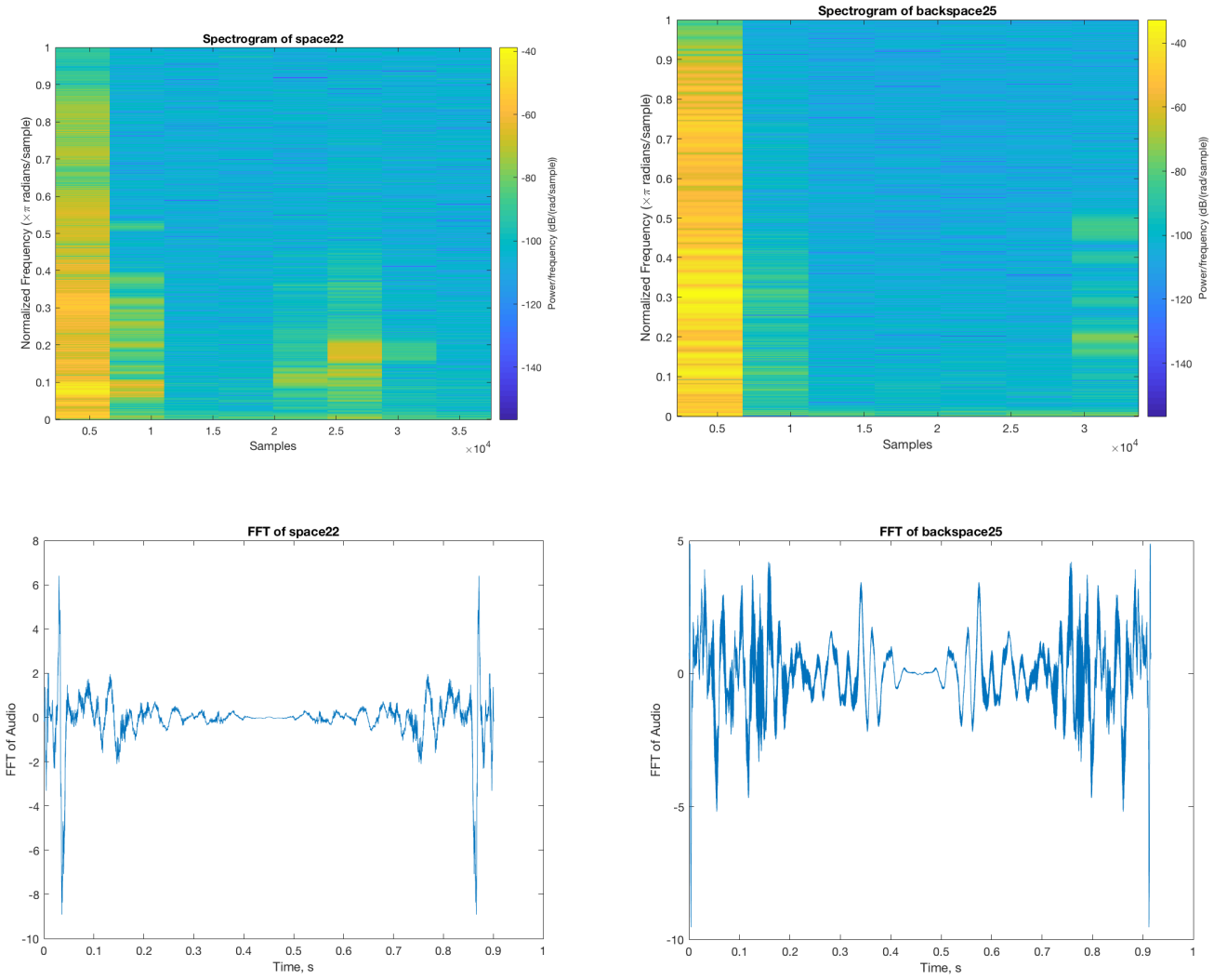
5/6/20:

Taylor:

Used google teachable machine to record and train a model containing 50 spacebar presses and 50 backspace presses. This teachable machine uses a fourier transform to eliminate background noises and can correctly identify the key pressed 60% of the time.

Harris:

A visualization of what the neural network is doing in the background to identifyspacebars versus backspaces:

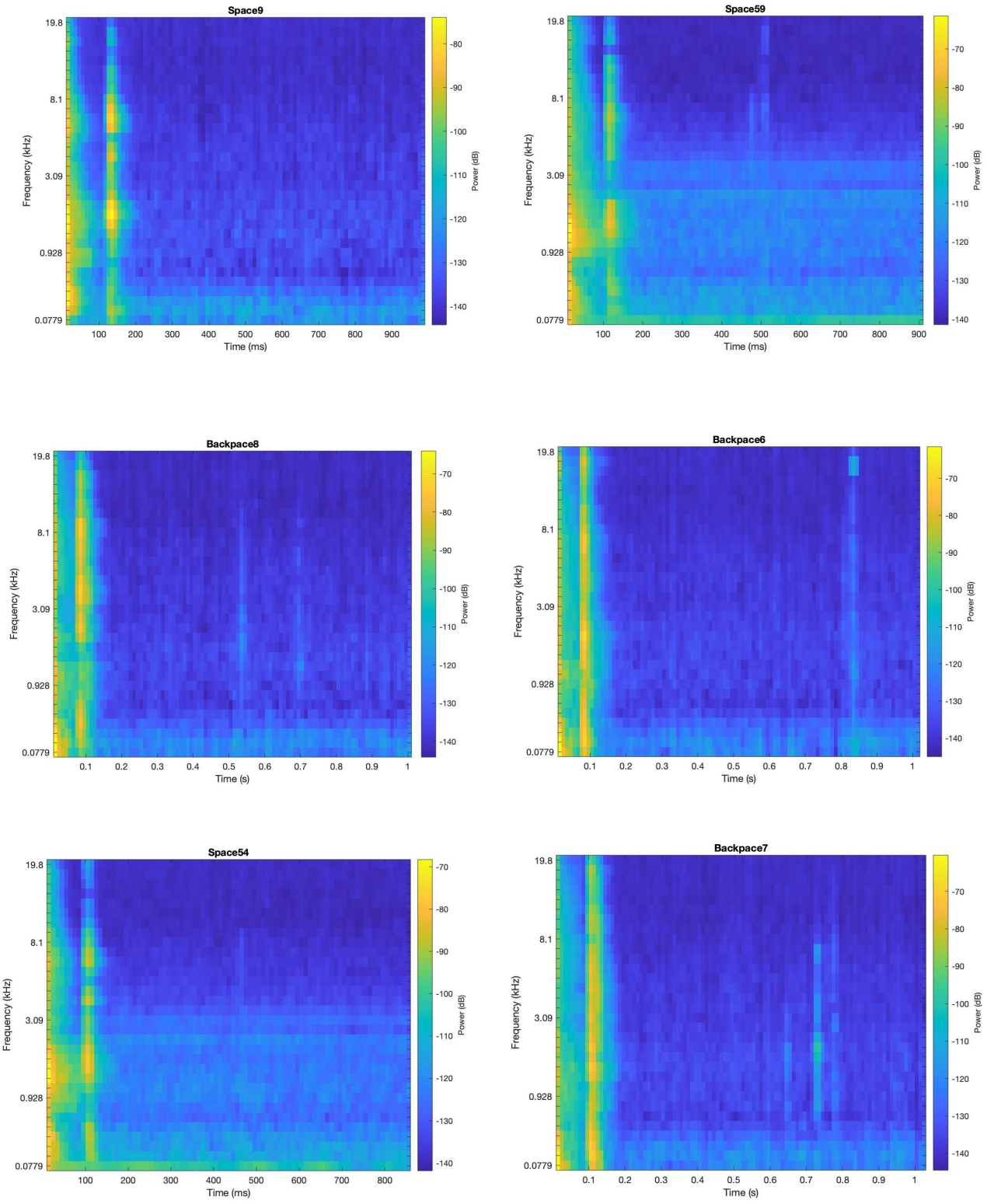


Figures 1-4: Fast Fourier Transform and Spectrogram of a space and backspace audio signal.

5/7/20:

Taylor:

Used MatLab to create a mel-spectrogram of three spacebars and three backspace key presses (Figures 1-6). These spectrograms show what Dan's tensorflow code is using to train the model.



Figures 1-6: Mel-spectrograms of spacebar and backspace keys

Dan:

Created a dataset containing a recording of the first row of keys on the keyboard being pressed and ran a model for this set. The fit accuracy for this model was 99% (using 256 data points) and the validation accuracy was only 33% (with 100 data points). We hypothesize that increasing the amount of data points will in turn increase the validation accuracy.

5/8/20:

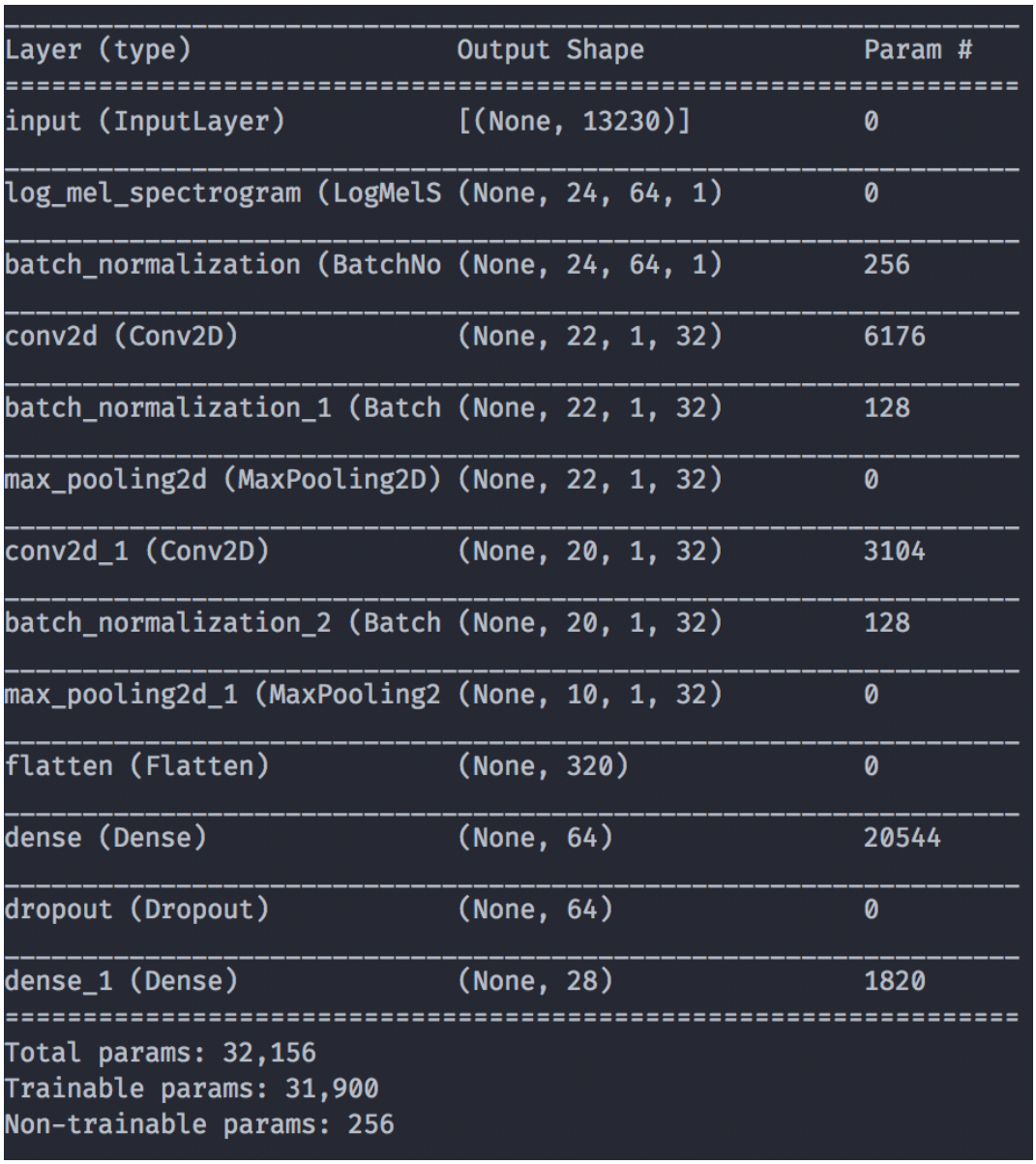
Due date for report, video, attribution document, and logbook.

Dan:

Created a model (code shown below) using random word typer tests from Taylor and Dan (10 datasets each) to create a model of the entire keyboard. When only using Dan's dataset the model fit accuracy (with epoch 100) was 98% (1500 data points) and the validation accuracy was 65% (200 data points). In order to validate this model we used 8 of the datasets for training and 2 for validation. The model fit accuracy for Taylor's data set is 99% and the validation accuracy was 55%. We think that the microphone on Taylor's laptop is more sensitive and therefore the data is more noisy. This resulted in a lower validation accuracy. After combining the datasets, the model fit accuracy was 99% and the validation accuracy was 60%.

Toprow and Fullkeyboard Model: adapted from Schewrtfeger

```
def ConvModel(n_classes, sample_rate=44100, duration=.3,
              fft_size=_FFT_SIZE, hop_size=_HOP_SIZE,
n_mels=_N_MEL_BINS):
    n_samples = sample_rate * duration

    # Accept raw audio data as input
    x = Input(shape=(int(n_samples),), name='input', dtype='float32')
    # Process into log-mel-spectrograms. (This is your custom layer!)
    y = LogMelSpectrogram(sample_rate, fft_size, hop_size, n_mels)(x)
    # Normalize data (on frequency axis)
    y = BatchNormalization(axis=2)(y)

    y = Conv2D(32, (3, n_mels), activation='relu')(y)
    y = BatchNormalization()(y)
    y = MaxPool2D((1, y.shape[2]))(y)

    y = Conv2D(32, (3, 1), activation='relu')(y)
    y = BatchNormalization()(y)
    y = MaxPool2D(pool_size=(2, 1))(y)

    y = Flatten()(y)
    y = Dense(64, activation='relu')(y)
    y = Dropout(0.25)(y)
    y = Dense(n_classes, activation='softmax')(y)

    return Model(inputs=x, outputs=y)
```



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (InputLayer)           [(None, 13230)]           0
_________________________________________________________________
log_mel_spectrogram (LogMelS (None, 24, 64, 1)         0
_________________________________________________________________
batch_normalization (BatchNo (None, 24, 64, 1)         256
_________________________________________________________________
conv2d (Conv2D)              (None, 22, 1, 32)         6176
_________________________________________________________________
batch_normalization_1 (Batch (None, 22, 1, 32)         128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 22, 1, 32)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 20, 1, 32)         3104
_________________________________________________________________
batch_normalization_2 (Batch (None, 20, 1, 32)         128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 10, 1, 32)         0
_________________________________________________________________
flatten (Flatten)            (None, 320)               0
_________________________________________________________________
dense (Dense)                (None, 64)                20544
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 28)                1820
=================================================================
Total params: 32,156
Trainable params: 31,900
Non-trainable params: 256
```

The above picture shows what is returned from running model.summary(). It shows, top to bottom (input to output), the different layers that are being run. The input shape has dimensions (None, 13230). None is a placeholder that basically holds batch information (how many samples are being fed into the model at a step per epoch). 13230 is the amount of samples given a sampling rate of 44.1 kHz over a time of .3 seconds. All of our data is normalized to .3 seconds due to testing and seeing that the average time for a key press and return to steady state (for Macbook Pro 2018+ butterfly keyboards) is around .2-.25 seconds, with .3 secs being used to

give us some headroom. If any sounds are too short, Tensorflow's audio.decode_wav function automatically pads the rest of the array with zeros in order to keep the amount of samples to 13230. When curating this final dataset, Taylor and Dan both maintained a typing speed of generally .35 seconds per letter. The second layer is the most important of all: the custom Log Mel Spectrogram layer that converts the raw audio sample to a spectrogram. The rest of the layers are a framework created by Schwertfeger that follows standard image processing. There are two rounds of convoluting and pooling, then a flattening layer that flattens all previous dimensions into one array, then outputs 28 nodes that represent a-z, spacebar and backspace.

Dan used the model to predict some of the training and validation data. This was to evaluate how the model can predict keys. We noticed that our overall keyboard model could predict spacebar and backspace very well, but overall it did not have as high of an accuracy as we had hoped for.

```
Due to how tensorflow parses features, I translated space, backspace, and a-z to numbers
Backspace: 0, Space: 1, a: 2, b: 3, c: 4, d: 5, ... , z: 27

Labels fed in: Results from model prediction: Formatted Results Below:

Input: 16, Prediction: 16
Input: 17, Prediction: 0
Input: 3, Prediction: 15
Input: 8, Prediction: 14
Input: 1, Prediction: 1
Input: 3, Prediction: 3
Input: 4, Prediction: 14
Input: 0, Prediction: 0
Input: 5, Prediction: 14
Input: 10, Prediction: 10
Input: 14, Prediction: 17
Overall Accuracy of Prediction at 11 inputs: 0.45454545454545453

Since prediction accuracy is skewed when using a small sample size, please refer to the actual accuracy presented by
the evaluator at 60% for this model
```

Due to the larger amount of space bars and backspaces within our dataset (roughly 25% spaces and 15% backspaces in the overall full keyboard dataset), our full keyboard model predicts spacebars and backspaces 100% of the time. This goes to show that if we trained our model on an even bigger dataset (100,000+ compared to 2800), we would have an equal representation of

all 28 classes and have a model that could predict other letters more reliably. At least we were able to get a validation accuracy of 60% and have a model that can reliably fit to training data with a very minimal loss.

Taylor:

Finished writing the report and helped analyze the model and data. Worked on the demo video.

Harris:

Finished writing the report and helped analyze the model and data. Worked on the demo video.

6.0 List of References:

Mendels, G. (2019, November 18). How to apply machine learning and deep learning methods to audio analysis. Retrieved April 17, 2020, from https://towardsdatascience.com/how-to-apply-machine-learning-and-deep-learning-methods-to-audio-analysis-615e286fcbbc

Schwertfeger, D. (2020, April 22). How to Build Efficient Audio-Data Pipelines with TensorFlow 2.0. Retrieved May 7, 2020, from https://towardsdatascience.com/how-to-build-efficient-audio-data-pipelines-with-tensorflow-2-0-b3133474c3c1

Schwertfeger, D. (2020, April 27). How to Easily Process Audio on Your GPU with TensorFlow. Retrieved May 7, 2020, from https://towardsdatascience.com/how-to-easily-process-audio-on-your-gpu-with-tensorflow-2d9d91360f06

Zhuang, L., Zhou, F., & Tygar, J. D. (2005). Keyboard acoustic emanations revisited. *Proceedings of the 12th ACM Conference on Computer and Communications Security - CCS '05*. doi:10.1145/1102120.1102169