8-bit Brain Final Report: Vocal Activity Detection and Noise Sensitivity

Dennis Brown
dgb0028@auburn.edu

Shannon McDade
slm0035@auburn.edu

Jacob Parmer
jdp0061@auburn.edu

1 Introduction

In many speech processing toolchains, the first building block of the system is vocal activity detection (VAD). A VAD algorithm categorizes the incoming audio data into human speech or noise to determine segments of the signal where there is a speaker present. In modern systems, this processing is performed by training a machine learning model to differentiate between the two states. Implementing this task into a speech processing system can help both the accuracy and performance by allowing it to focus exclusively on the audio data that is important to the application.

These new models for speech processing are commonly implemented as various forms of Neural Networks, which have been shown to outperform statistical models such as the Gaussian Mixture Model. [6] For our project, we intended to use a Recurrent Neural Network (RNN); however, we encountered unexpected difficulties with our original model during the execution of the project. Ultimately, due to time constraints we decided to modify, test, and analyze an existing 1-D Residual Network, or Resnet, model by Filippo Giruzzi. [3]

The test-clean subset of the LibriSpeech data set was used as our input data. [7] Moreover, we added varying, increasing levels of noise to the test-clean data in order to analyze the performance of the VAD as noise levels increase.

2 Literature Review

2.1 Related Work

As VAD algorithms have become an integral part of many speech processing applications, there is a significant body of knowledge already available on this subject. The research and implementations shown below provided the groundwork for our project:

- Hansen and Albrechtsen show that convolutional and recurrent neural networks are viable for VAD in noisy environments. [5]
- Bäckström explains shortcomings of non-machine learning attempts at VAD, and briefly examines the steps necessary to implement an effective VAD with machine learning (ML). [1]
- Giruzzi provides his ML implementation, and steps on how to run it, using Deep Learning and TensorFlow. [3]

3 Task Definition

3.1 Problem Scope

For our project, we set out to train and test a VAD with varying, increasing levels of noise in our dataset. In doing this, we could evaluate the resiliency of the model against more "real-world" scenarios where the audio would not be entirely clean. This effort also could help prove the superiority of a machine learning based approach as compared to something like a simple "noise gate" which would filter out any portion of a signal beneath a certain volume. Ideally, our model would be able to differentiate voiced and unvoiced sections regardless of noise levels present in the signal.

3.2 Evaluation

As this was a supervised learning task, our evaluation was completed by comparing labels, which were provided by Giruzzi, to our outputs. These labels specified positions in the array of floating point values that represented the audio signal at which speech started or stopped. An accuracy was derived from this using Tensorflow's accuracy metrics for each of the varying noise models described in section 3.3.2. Figure 1 shows an visual example of an input signal with speech areas in green and noise areas in red.

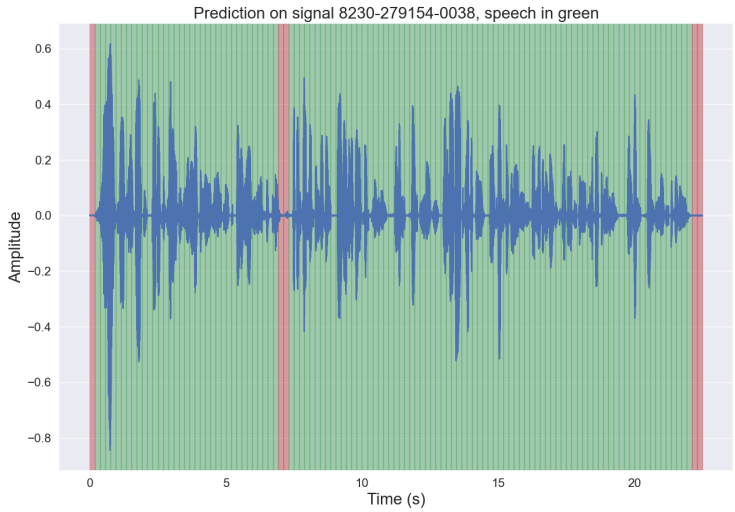

Figure 1: Visualized prediction of speech vs. non-speech data

3.3 Data Source

3.3.1 Original Data

This project utilized the LibriSpeech ASR Corpus data set. [7] This data set is a commonly used data collection featuring approximately 1000 hours of English speech from read-aloud audiobooks from the LibriVox project. For our approach, we used the `test-clean` subset of the data, and split it into training (70%), validation (15%), and testing (15%) sets. While `test-clean` was only a small sub-sample of the total LibriSpeech data set, it was the only data set that came pre-labeled and it proved to be large enough to train a model reliably enough for our purposes.

3.3.2 Data Modification

A goal of our project was to find out how well would this implementation would perform training and testing on noisy models, in addition to how well it would perform training at one level of noise and testing at another level of noise. In order to accomplish this goal, we had to add noise to the original test-clean data set. The test-clean data set contains 2,620 Free Lossless Audio Codec (FLAC) audio

files aka *signals*, which include labels of what is speech and what is noise. The average signal is about 8.2 seconds long with a loudness of approximately -22.5 decibels relative to full scale (dBFS).

We replicated the test-clean data set seven additional times, adding white noise using custom code and the `pyDub` library (see `util/noisemaker.py` in the code repository). Each data set matched the original's variable bit rate, 16.0 kHz, 16 bits, and lossless compression. We added noise to these data sets at these levels of loudness: -160, -80, -40, -20, -10, -5, and 0 (maximum).

3.4 Infrastructure

Throughout this project, we relied on several existing tools. This included both PyTorch for our original implementation and TensorFlow for our modified implementation. These tools allowed us to quickly build and use powerful neural networks. We also utilized the LibriSpeech dataset, which we discussed in more detail in section 3.3. Some libraries we relied on, which are commonly used in machine learning projects, incuded NumPy, pandas, matplotlib, plotly, and librosa.

3.5 Challenges

In building this design, we anticipated certain problems would arise. A significant hurdle for us was a lack of information regarding implementation during our research phase. This situation made a lot of our personal implementation a lot of trial-and-error, which resulted in us not finishing this aspect of the project in the limited time frame given to us for this semester. Additionally, since we were running our algorithms on non-optimal hardware, training and testing for our models took a significant amount of time. However, since we started relatively early, this didn't end up being a huge issue.

4 Approach

As previously mentioned, we had two approaches for this project. Firstly, our own implementation that was incomplete, and also a modified implementation of existing code that we used to collect data.

4.1 Original Implementation

Originally, we intended to fully build our own VAD. The flowchart for our intended implementation is shown in Figure 2. However, due to time constraints, this implementation was put aside so that we could focus on getting meaningful results through another approach. Nonetheless, we believe it is still worth mentioning and describing our implementation here, as much of this work is still significant and relevant to our goals for this project.

In this design, the goal was to train a Recurrent Neural Network (RNN) model on the LibriSpeech dataset described in section 3.3 using PyTorch. Rather than passing in the full audio signal data as the features for this model, it is typical to extract information from the signal, and pass this information in instead. [4] For our model, we were using the *Mel Filterbank* as our input feature. This representation was chosen due to ease of implementation, as the Mel Filterbank has a constant size regardless of the length of the original input audio signal, which makes feeding it into a neural network possible without any pre-processing. An example filterbank from our implementation can be seen in Figure 3

Unfortunately, the RNN model we attempted proved to take a considerable amount of time, which is what prompted us to focus on a different avenue.

4.2 Modified Implementation

In order to obtain more meaningful results, we modified the existing implementation provided by Giruzzi [3] so that we could train the data on varying degrees of noise and verify the accuracy of our predictions. Our modifications to this implementation are clearly marked in the source files in the directory `vad` in our code repository.

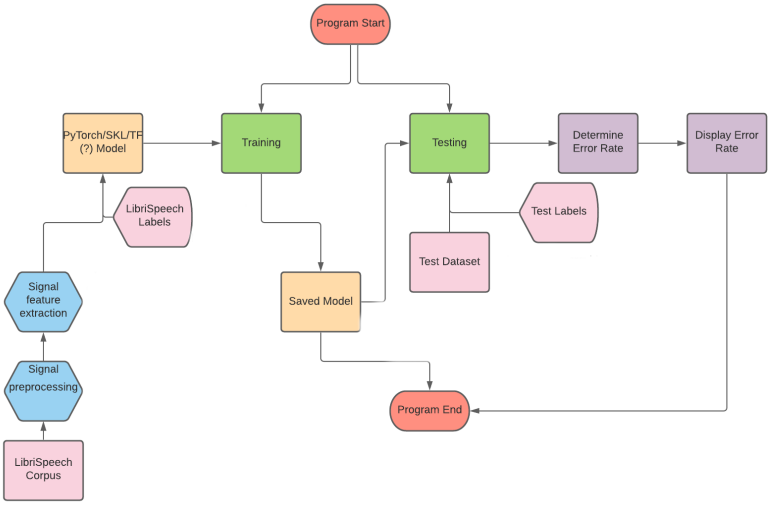

Figure 2: Flowchart of intended VAD program execution

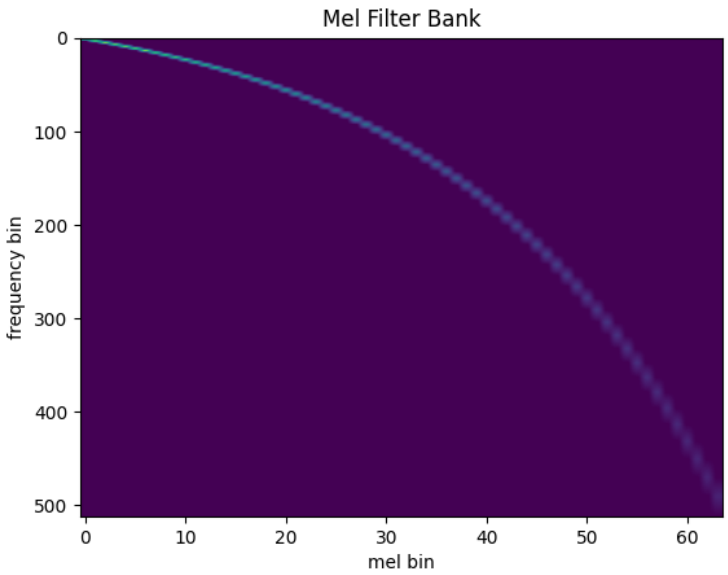

Figure 3: Mel filterbank, separating frequencies from the signal into bins in the mel scale [2]

The Giruzzi implementation uses ResNet1d, which consists of four Convolutional Neural Network (CNN) blocks plus additional layers built with TensorFlow's Conv1d, BatchNormalization, Activation, Dense, and Flatten. It was configured as follows:

- Sample (1024 features) ⇒
- Block 1:
  - 1-D Conv1d(32 filters, 8 kernels) ⇒ BatchNormalization ⇒ Activation(ReLU) ⇒
  - 1-D Conv1d(32 filters, 5 kernels) ⇒ BatchNormalization ⇒ Activation(ReLU) ⇒
  - 1-D Conv1d(32 filters, 3 kernels) ⇒ BatchNormalization ⇒
- Block 2:

  - 1-D Conv1d(64 filters, 8 kernels) ⇒ BatchNormalization ⇒ Activation(ReLU) ⇒
  - 1-D Conv1d(64 filters, 5 kernels) ⇒ BatchNormalization ⇒ Activation(ReLU) ⇒
  - 1-D Conv1d(64 filters, 3 kernels) ⇒ BatchNormalization ⇒
- Block 3:
  - 1-D Conv1d(128 filters, 8 kernels) ⇒ BatchNormalization ⇒ Activation(ReLU) ⇒
  - 1-D Conv1d(128 filters, 5 kernels) ⇒ BatchNormalization ⇒ Activation(ReLU) ⇒
  - 1-D Conv1d(128 filters, 3 kernels) ⇒ BatchNormalization ⇒
- Block 4:
  - 1-D Conv1d(128 filters, 8 kernels) ⇒ BatchNormalization ⇒ Activation(ReLU) ⇒
  - 1-D Conv1d(128 filters, 5 kernels) ⇒ BatchNormalization ⇒ Activation(ReLU) ⇒
  - 1-D Conv1d(128 filters, 3 kernels) ⇒ BatchNormalization ⇒
- Flatten ⇒
- Dense (Fully-Connected layer of 2048 units) ⇒
- Dense (Fully-Connected layer of 2048 units) ⇒
- Dense (Fully-Connected layer of 2 units) ⇒
- Classification of sample as Speech or Noise

5 Experimental Methodology

Using the modified Giruzzi implementation, we first converted the 2,260 raw audio data signals (FLAC files) into TensorFlow records by extracting Mel Frequency Cepstrum Coefficients (MFCC) from the audio files to use as the feature data with 1024 dimensions per sample. Figure 4 shows a visualization of MFCCs.

With the converted data, we trained a ResNet1d instance as described in section 4.2 for each of the 8 data sets (original plus these levels of white noise added: -160 dBFS, -80 dBFS, -40 dBFS, -20 dBFS, -10 dBFS, -5 dBFS, and 0 dBFS (maximum loudness). This training used a batch size of 32 performed over 20 epochs. Each model was trained on 70% of the data set (1834 samples) and required approximately 210 minutes of computing time on a high-end laptop.

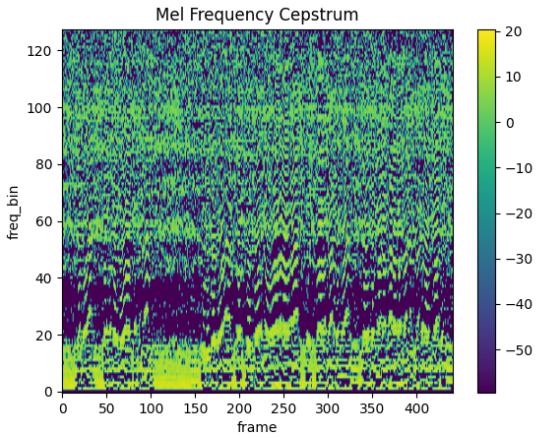

Figure 4: Visualization of Mel Frequency Cepstrum Coefficients (MFCCs)

Next, we tested the eight models on each of the eight data sets on the 15% of the data set (393 samples) we reserved for testing. We modified the Giruzzi implementation to collect and report statistics about accuracy sufficient to build confusion matrices. Each test required approximately nine minutes of computing time, and consequently, all 64 tests required approximately 576 minutes.

In the code repository, `README.md` explains how to set up and reproduce this experiment.

6 Results and Analysis

First, we look at the raw accuracy data. Figure 5 shows the accuracy of testing all the models (across the top row) against all the data sets (down the left side). We spot-checked cases where the numbers are close and found that for most cases within 2% of each other, there is no statistical difference in accuracy per F- and t-testing when looking at accuracy per sample. In the interest of saving space we have not included the tables showing mean, variance, degrees of freedom, etc. for each of those tests, but the results of the analysis can be found in the repository under `results/compare_results.txt` and those tables can be generated on-demand using `util/resultsCompare.py`.

model / dataset	orig	db-160	db-80	db-40	db-20	db-10	db-5	db0
orig	97.36	97.91	97.67	89.03	78.58	76.67	55.2	67.48
db-160	97.06	97.88	97.44	88.88	78.15	75.7	54.68	68.4
db-80	96.98	97.85	97.49	89.54	78.98	76.79	54.55	69.14
db-40	86.48	90.15	86.68	96.55	92.01	84.57	39.62	72.42
db-20	75.12	77.16	75.26	76.82	79.35	65.76	30.96	58.76
db-10	77.7	77.7	77.7	77.7	77.06	64.28	34.46	50.43
db-5	77.7	77.7	77.7	77.7	77.7	77.7	54.79	76.33
db0	77.7	77.7	77.7	77.7	77.7	77.7	77.52	58.77

Figure 5: Accuracy

As expected, the strongest performance is in the upper left of the accuracy table (5) where no- and low-noise models performed well against no- and low-noise data sets. One may also expect strong performance along the diagonal where models were tested against the (reserved test portion of the) data sets on which they trained. However, there are two anomalies that stick out. First, one will observe along the bottom-left of the table a consistent 77.7% accuracy of models trains on low noise tested on very noisy data sets. Second, one will observe an anomalous-appearing "hotspot" in the -5 dBFS model column.

In investigating these results, we will look at confusion matrix data, where the confusion matrix conceptually consists of Predicted Speech and Predicted Noise versus Actual Speech and Actual Noise. Since we are analyzing 64 test combinations, we will split apart the traditional confusion matrix into four separate tables. First, we consider Predicted Speech versus Actual Speech shown in Figure 6. It shows even better performance of predicting Speech in those bottom-left cells.

model / dataset	orig	db-160	db-80	db-40	db-20	db-10	db-5	db0
orig	0.77	0.78	0.77	0.79	0.79	0.71	0.33	0.67
db-160	0.76	0.77	0.77	0.79	0.79	0.71	0.33	0.69
db-80	0.76	0.77	0.77	0.79	0.79	0.71	0.33	0.69
db-40	0.65	0.69	0.65	0.76	0.76	0.64	0.17	0.6
db-20	0.73	0.78	0.75	0.75	0.6	0.45	0.09	0.45
db-10	0.79	0.79	0.79	0.79	0.77	0.51	0.15	0.36
db-5	0.79	0.79	0.79	0.79	0.79	0.79	0.42	0.76
db0	0.79	0.79	0.79	0.79	0.79	0.79	0.79	0.49

Figure 6: Predicted Speech versus Actual Speech

However, if we look at Predicted Speech versus Actual Noise in Figure 7, we see the rest of the story for those bottom cells. The less-noisy models *always* predict Speech when tested on noisy data sets, and the speech files seem to have an average of 77.7% speech versus noise. While the low-noise models appear to perform well on the noisy data in simple accuracy, the confusion matrix shows they are actually very poor at predicting noise, which makes sense because they were not trained on noise. These low-noise models would not be suitable for predicting speech in high-noise data sets, but since they default to "Speech," they may give a suitable failure mode in applications that need to conservatively detect speech. Next, we consider the "hotspot" under -5 dBFS.

model / dataset	orig	db-160	db-80	db-40	db-20	db-10	db-5	db0
orig	0	0.01	0.01	0.11	0.2	0.15	0	0.2
db-160	0	0.01	0.01	0.11	0.21	0.16	0	0.2
db-80	0	0.01	0	0.1	0.2	0.15	0	0.2
db-40	0	0.01	0	0.01	0.05	0.01	0	0.09
db-20	0.18	0.21	0.2	0.18	0.02	0.01	0.01	0.08
db-10	0.21	0.21	0.21	0.21	0.2	0.08	0.03	0.08
db-5	0.21	0.21	0.21	0.21	0.21	0.21	0.08	0.2
db0	0.21	0.21	0.21	0.21	0.21	0.21	0.21	0.11

Figure 7: Predicted Speech versus Actual Noise

Looking at Predicted Noise versus Actual Speech in Figure 8, we see the model for -5 dBFS, which trained on samples that were almost all noise, is too eager to predict noise on the less-noisy data sets and thus displays the poor performance shown.

model / dataset	orig	db-160	db-80	db-40	db-20	db-10	db-5	db0
orig	0.02	0.01	0.02	0	0	0.08	0.46	0.12
db-160	0.03	0.02	0.02	0	0	0.08	0.46	0.1
db-80	0.03	0.02	0.02	0	0	0.08	0.46	0.1
db-40	0.14	0.1	0.14	0.03	0.03	0.15	0.62	0.18
db-20	0.06	0.01	0.04	0.04	0.19	0.34	0.7	0.34
db-10	0	0	0	0	0.02	0.28	0.64	0.43
db-5	0	0	0	0	0	0	0.37	0.03
db0	0	0	0	0	0	0	0	0.3

Figure 8: Predicted Noise versus Actual Speech

For completeness, Figure 9 shows Predicted Noise versus Actual Noise. It simply corroborates findings described previously.

model / dataset	orig	db-160	db-80	db-40	db-20	db-10	db-5	db0
orig	0.21	0.2	0.2	0.1	0.01	0.06	0.21	0.01
db-160	0.21	0.2	0.21	0.1	0	0.05	0.21	0.01
db-80	0.21	0.2	0.21	0.11	0.01	0.06	0.21	0.02
db-40	0.21	0.21	0.21	0.2	0.16	0.2	0.21	0.12
db-20	0.03	0	0.01	0.03	0.19	0.2	0.2	0.13
db-10	0	0	0	0	0.01	0.13	0.18	0.13
db-5	0	0	0	0	0	0	0.13	0.01
db0	0	0	0	0	0	0	0	0.1

Figure 9: Predicted Noise versus Actual Noise

Putting it all together: Figures 10 and 11 show two angles of a 3D representation of all accuracy and confusion data for all combinations of model and data set in as meshes. The layers have the best separation and best performance in the low noise region.

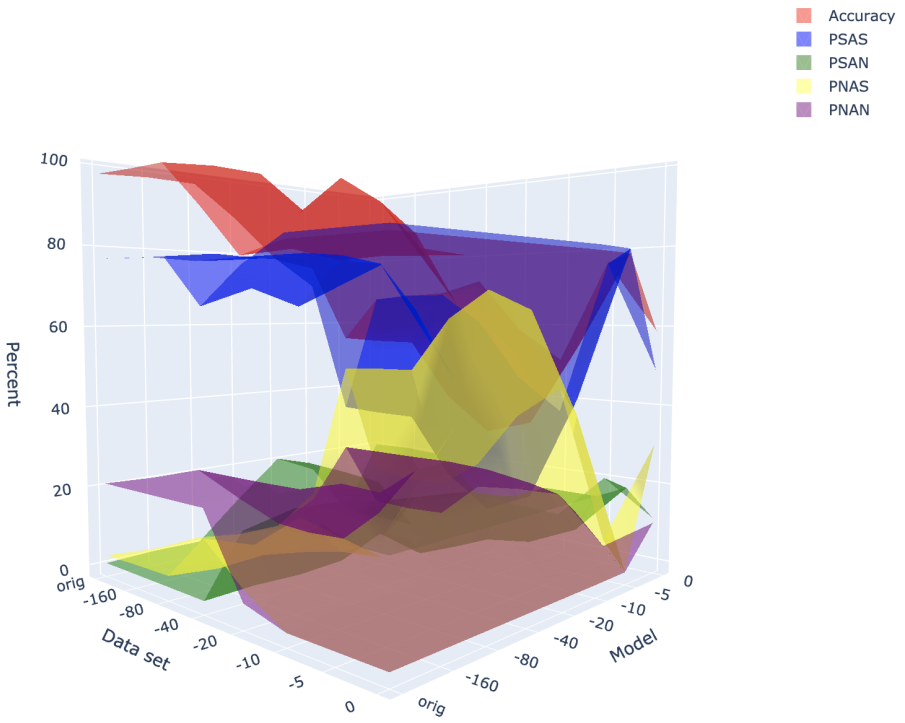

Figure 10: Visualization of accuracy and confusion data

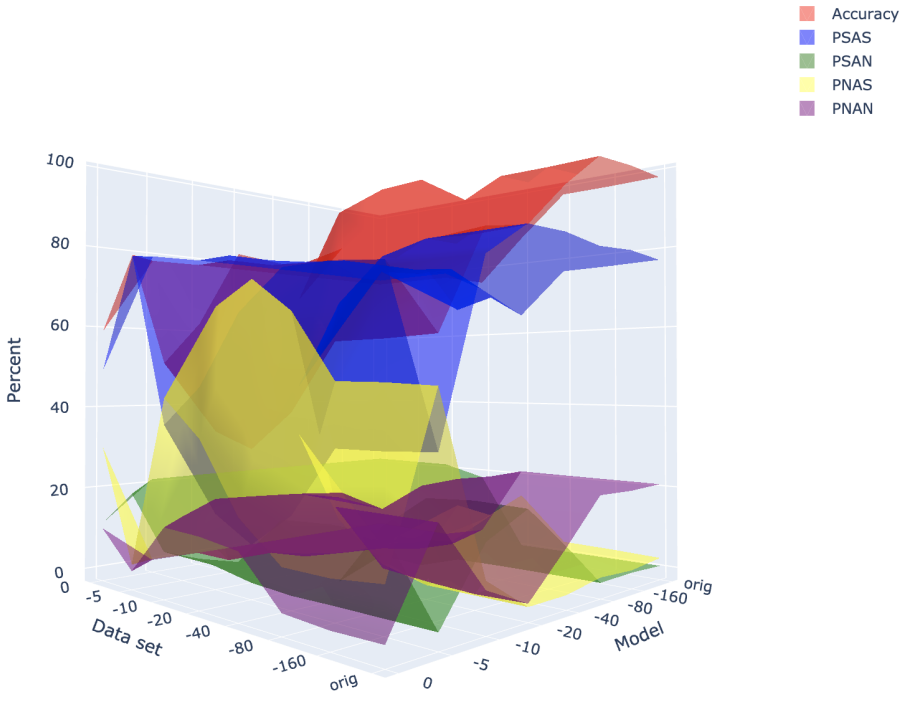

Figure 11: Visualization of accuracy and confusion data, reverse view

7 Conclusion

From the above data, we came to the following conclusions. Firstly, our models performed better in no- or low-noise environments, as expected. The models performed particularly poorly for extremely high levels of noise, at which point the network was unable to separate the noise from the speech effectively. Thus, we conclude that a vocal activity detector on pure noisy data might not be the most effective approach. This task could perhaps be improved upon by adding some pre-processing steps to "de-noise" the signal, or potentially by selecting an alternate input feature. Also, since we saw a slight uptick in performance on the -160dB model (very low level of noise), it may be worthwhile to train models on very small levels of noise, as opposed to completely clean audio data.

References

[1] Tom Bäckström. *Voice Activity Detection (VAD)*, 2020 (accessed Feb 28, 2021). https://wiki.aalto.fi/pages/viewpage.action?pageId=151500905.

[2] Nikos Drakos. *Filterbank Analysis*, 1997 (accessed Apr 16, 2021). https://labrosa.ee.columbia.edu/doc/HTKBook21/node54.html.

[3] Filippo Giruzzi. *Voice Activity Detection based on Deep Learning & TensorFlow*, 2020 (accessed Mar 4, 2021). https://pythonawesome.com/voice-activity-detection-based-on-deep-learning-tensorflow/.

[4] Simon Graf, Tobias Herbig, Markus Buck, and Gerhard Schmidt. *Features for voice activity detection: a comparative analysis*, 2015 (accessed Apr 15, 2021). https://link.springer.com/article/10.1186/s13634-015-0277-z.

[5] Nicklas Hansen and Simon Holst Albrechtsen. Voice activity detection in noisy environments. December 2018.

[6] Thad Hughes and Keir Mierle. *RECURRENT NEURAL NETWORKS FOR VOICE ACTIVITY DETECTION*, 2013 (accessed Apr 16, 2021). https://storage.googleapis.com/pub-tools-public-publication-data/pdf/41186.pdf.

[7] Daniel Povey, Vassil Panayotov, Guoguo Chen, and Sanjeev Khudanpur. *LibriSpeech ASR corpus*, 2015 (accessed Feb 22, 2021). https://www.openslr.org/12/.