ADVANCED AUDIO FILTERING FOR R21 USING MACHINE LEARNING

By

SAM JARED GIANELLI

____

A Thesis Submitted to The Honors College

In Partial Fulfillment of the Bachelors degree
With Honors in

Electrical and Computer Engineering

THE UNIVERSITY OF ARIZONA

M A Y 2 0 1 8

Approved by:

____

Gary Redford
College of Engineering

Abstract

The goal: To expedite cleaning of maritime distress signals without manual filtering.

Maritime distress transmissions received by the U.S. Coast Guard are subject to environmental noise and communication channel distortion, and must be filtered manually using current Rescue 21 software, which can be time consuming and imprecise.

This project uses deep learning to improve signal filtration. The core of the software is a de-noising auto-encoder, a machine learning model that recognizes and removes distortion from input audio files. Noisy distress signals are input into the team's software and automatically filtered by the auto-encoder, which generates a coherent, noiseless version of the input signal.

Testing required the software to process a 5-minute audio segment in under a minute while using less than 2 GB of local memory. The software is compatible with all operating systems used by the Coast Guard, including the existing Rescue 21 software.

Content Disclaimer

The research and information contained herein does not include any Rescue 21 software, data, or other direct Project material.

Statement of Roles & Responsibilities

Alexander Brooks	Led the creation of the Machine Learning component of the project. Conducted most of the background research, literature reviews, and programming that led to our eventual finalized design.
Sam Gianelli	Team leader and primary liaison between the team and the project sponsors (General Dynamics). Also contributed to the programming and training of the Machine Learning component of the project.
Michael Ragone	Led the creation of the audio conversion process. Conducted all research, made most technical decisions, and programmed a large majority of the audio to Mel Frequency Cepstral Coefficients conversion.
Jodi Rickel	Led the documentation process of all project code to ensure its replicability when passed to General Dynamics. Also led the testing and coordination of all project components to ensure that all requirements were met.
Patrick Wegner	Led the data collection for the Machine Learning component of the project. Responsible for the generation of the majority of data used to train our finalized model and contributed code to the audio conversion process.

Advanced Voice Filtering for R21 using Machine Learning
Final Report

Document Number 101, 700

Team 17053
Sam Gianelli, Patrick Wegner, Alexander Brooks, Jodi Rickel, Michael Ragone

March 26 2018 rev (-)
Prepared for General Dynamics

Table of Contents

1.0 Scope

This final report is for a new Advanced Voice Filtering strategy for Rescue 21 (R21) utilizing Machine Learning for improved isolation of vocal audio from maritime distress calls. The improved R21 system employs machine learning tools to automatically recognize distress calls and robustly clean them. Such a solution will be autonomous and more rapid than the current manual system, allowing the Coast Guard to more quickly respond to distress calls, hopefully saving more lives in the process.

This solution is unique and novel in the fact that our aim is to implement a fully automated system wherein audio filtration does not need to be conducted manually to achieve optimal results. With a working system such as this, human technicians would not be required to manually filter these signals and these signals would be filtered faster than before with said human technicians.

This document demonstrates our finalized design and the results of our Acceptance Tests and Models. In addition, this documents shows our final budget and lessons learned from this project.

1.1 Indentured Document List

Part Number	Document
101,000	**R21 Machine Learning Top Assembly**
100,100	**R21 Machine Learning Top SDD**
100,110	File Input/Output SDD
100,120	.Wav Splitter/Splicer SDD
100,130	MFCC Converter SDD
100,140	Neural Network SDD
100,200	**System Models**
100,210	Memory Usage Model
100,220	Cost Model
100,300	**Acceptance Test Procedure (ATP)**
101, 400	**System Requirement Document**
101, 500	**Test Plan**
101, 600	**EVMS**
101, 700	**Final Report**

Table 1, Indentured Document List

1.2 Statement of Work

Work for this project will be split into four phases based on important milestones.The first phase includes all activities up to and including the proposal. This includes the draft proposal, creation of a Concept of Operations (ConOps), and necessary research for the project. Milestones during this phase include the completion and sign-off on the proposal by 9/12/2017.

Phase 2 includes all activities up to and including the completion of the Preliminary Design Review (PDR). This includes continuing project research with using the trades and point design, gathering requirements and tolerances, creating the Systems Requirements Document (SRD) , and designing the architecture. Milestones during this phase include the completion and presentation of the PDR by 11/2/2017.

Phase 3 includes all activities up to and including the completion of the Critical Design Review (CDR). This includes creating the Technical Data Package (TDP), the completion of software design drawings, writing the Software Design Document (SDD), and designing the Acceptance Test Procedure (ATP). Milestones during this phase include the completion and presentation of the CDR by 12/5/2017.

Phase 4 includes all activities up to and including the final report. This includes the code development and test verification of the product, creation of the Final Acceptance Review (FAR), and the completion of the Customer Satisfaction (CS) report for General Dynamics. Milestones during this phase include the completion of the FAR by 4/24/2018, the design day presentation 4/30/2018, and the completion of the Final Report by 5/3/2018.

2.0 System Block Diagram

The internal structure of the machine learning model can be decomposed as shown.

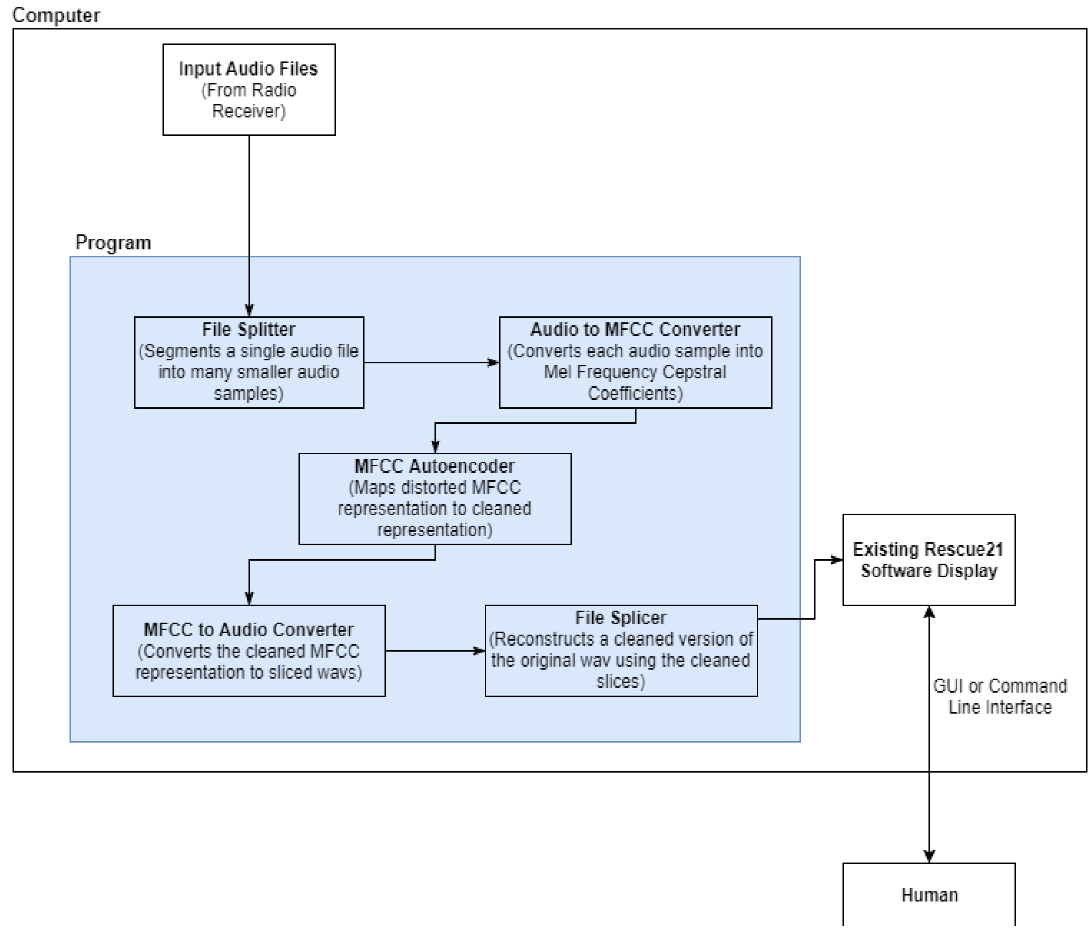


Figure 1, System Decomposition

The primary Interface for the system consists of the bridge between input audio files to be cleaned and the file splitter. The output files of the system will be automatically saved to a project directory which can be integrated into existing R21 software by General Dynamics at a later time.

Sub-Assemblies

Standalone verifiable subassemblies consist of the five blocks defined in the blue program block. The flow of the program is to automatically partition a file into chunks, convert audio chunks into MFCC representation, clean the MFCCs using a trained autoencoder, convert the result back to audio chunks, splice the cleaned audio chunks together, and save the result to a directory. The roles of the subassemblies are directly defined below.

- File splitter - Breaks an audio file into chunks
- Audio to MFCC Converter - Converts audio chunks to MFCC representation
- MFCC Autoencoder - Cleans MFCC using a trained deep neural network
- MFCC to Audio Converter - Converts (cleaned) MFCC chunks to audio
- File Splicer - Stitches cleaned audio chunks back together to produce cleaned sample

Interfaces

Because this project is entirely software implementation, there are no distinct interfaces between subassemblies. Each subassembly essentially comprises a main class or function which will be called within a main wrapper program. As such, USBs, wiring, attachments, etc. will not be discussed within the TDP. The only requirement for our software to run is a functioning computer on which the software can be deployed (either as Python scripts or as a standalone executable). The only “interface” we will define for this project is how the user interacts with this software. This interaction will be through the command line and is discussed in detail in the SDD (Section 3).

3.0 Technical Data Package

Advanced Voice Filtering for R21 using Machine Learning
Software Design Document

Document Number 100,100

Team 17053
Sam Gianelli, Patrick Wegner, Alexander Brooks, Jodi Rickel, Michael Ragone

January 23 2018 rev (-)
Prepared for General Dynamics

Table of Contents

3.1 Scope

This Advanced Voice Filtering for R21 using Machine Learning SDD gives the design of the software CSCI for the Rescue 21 system, document 101,000. The purpose of this software is to implement a fully functioning Neural Network as a means to use Machine Learning to automatically denoise and cleanly filter noisy maritime distress signals. To achieve this, the software must be able to acknowledge and read noisy signals as input, preprocess this data such that it is transformed into a usable format, process the data in our neural network, then save the output as a sound file from which the Coast Guard can interact with and listen to.

The software shall meet all requirements stated within the system requirement matrix as well as the requirements necessitated by the architectural design. The software shall significantly clean distorted audio signals. The software will be developed using Amazon Web Services (AWS) products including Elastic Cloud Computing (EC2) resources as well as Simple Storage Service (S3) for data storage. The Software shall execute in a time efficient and memory efficient manner. The software shall satisfy all of these requirements. For an in depth look at the requirements and verification strategies, consult section 2 of this Technical Data Package.

Document Overview

This document defines all software modules, functions, and interfaces required to automatically filter noisy distress signals using a machine learning framework. Relevant CSCs (Computer Software Components) include:

1. File Input/Output
2. Wav Splitter/Splicer
3. Mel Frequency Cepstral Coefficient (MFCC)/Wav Converter/Inverter
4. Autoencoder Neural Network.

3.2 Reference Documents

2012-001, Recurrent Neural Networks for Noise Reduction in Robust ASR, Rev A
2017-001, Amazon Elastic Compute Cloud - User Guide, Rev A
2017-002, Getting Started with Tensorflow, Rev A

3.3 CSCI Design Decisions

File (pre/post) processing CSCI Design Decisions

The CSCI inputs shall include one or more wav files. The outputs shall include a trained autoencoder (training mode) or a cleaned set of corresponding wav files (deployment mode).

File processing modules shall make use of Scipy and Numpy, which are Python libraries commonly used for numerical computing on signals. Files shall be loaded in with Scipy, and processed with Numpy. The result will be passed through the machine learning model. The model output will undergo a series of inverse Numpy operations, the result of which shall be saved with Scipy. Note that we will also include our scripts to train the network (not necessary to deploy the existing model, but required for generalizing to different datasets), which will optionally assess performance using Numpy and Tensorflow.

User Interaction Design Decisions

The system shall not contain a GUI, but shall provide a verbose flag for optional command line output, giving relative timelines for operation completion. If the verbose flag is disabled, information will be piped to log files in the current working directory.

Machine learning model Design Decisions

The CSCI shall utilize the deep denoising neural network autoencoder architecture proposed in reference document 2012-001, implemented in Tensorflow, and trained on an Amazon Elastic Compute Cloud Instance with the maximum batch size permitted by memory bandwidth. The preliminary model will be trained with a learning rate of 0.0001 using Stochastic Gradient Descent, however this is subject to change. After experimentation, the parameters deemed best to train our model were a learning rate of 0.01 using Momentum as our optimization function.

3.4 CSCI Architectural Design

Below are the CSCs (primary modules and functions) that will be implemented within our our CSCI as well as how they are related. The logic and routines given for each CSC are to detail the primary design principles to validate our design. We will employ some pre-made libraries to aid in our design process and expedite the design process.

Libraries:

- *Scipy:* File processing library with capacity for handling .wav files
- *Numpy:* Mathematics library for Python
- *Tensorflow:* Deep Learning / Machine learning library and framework

CSC	Characteristics/Relationship	Purpose	Planned Resource	Library
File Input/Output	Takes in wav files and inserts them into the splitter, also exports wav files from splicer	Receives noisy signal(s) and outputs denoised signal(s)	5MB	Scipy
File Splitter/Splicer	Splits input wav into smaller duration wav files for MFCC Converter. Splices small wav into full length file and outputs to file I/O	Segments input files into smaller fragments for analysis. Stitches file fragments into one audio file for listening	5MB	Scipy
MFCC Converter/ Inverter	Receives small wav files from splitter, outputs MFCC to Neural Net. Receives MFCC from Neural Net, outputs small wav to splicer	Converts audio samples into MFCC vectors. Reconstructs audio chunks from MFCC vectors	10MB	Numpy
Neural Network Autoencoder	Receives small MFCC from converter, outputs small MFCC vector to inverter	Filters noise from input MFCC vectors and produces output MFCC vectors	70MB	Tensorflow

Table 2, Libraries used in Software

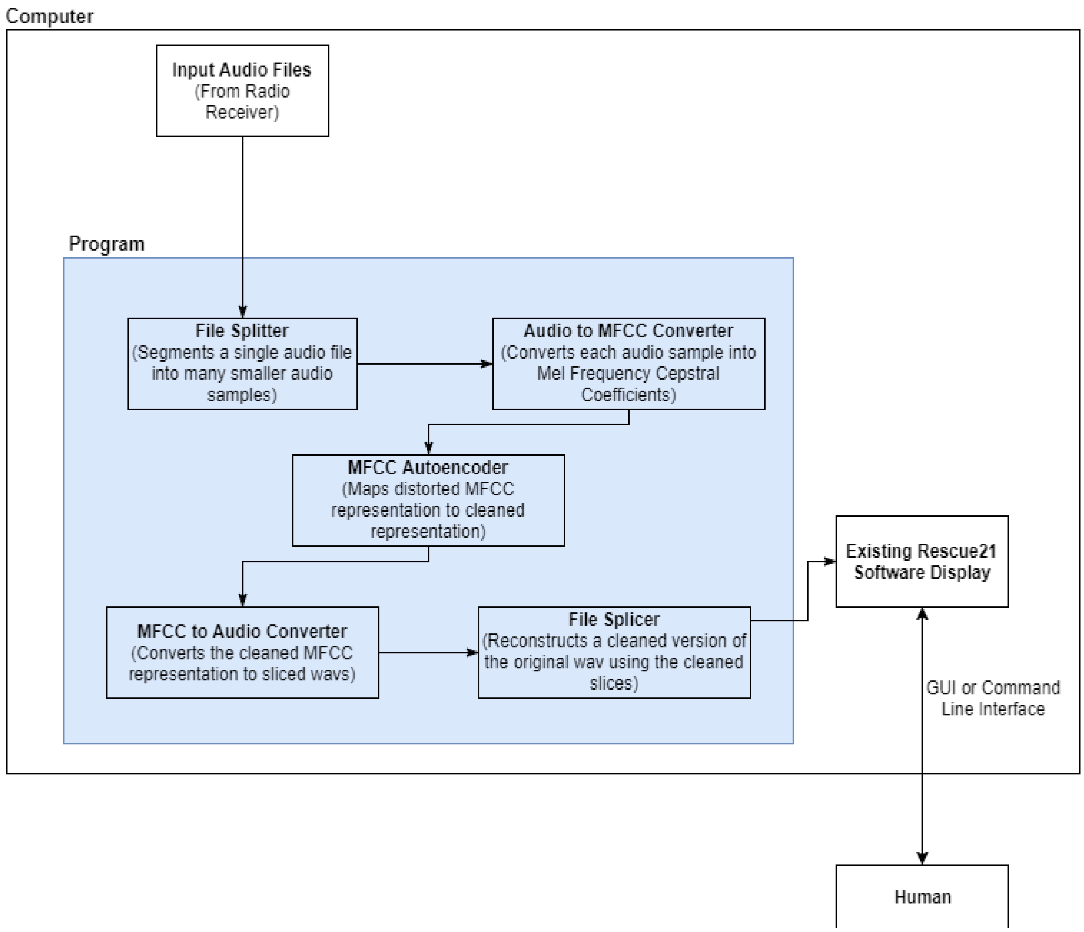


Figure 7, Advanced Voice Filtering for R21 using Machine Learning interfaces and Flow Chart. Describes the interactions between all CSCs in our main CSCI.

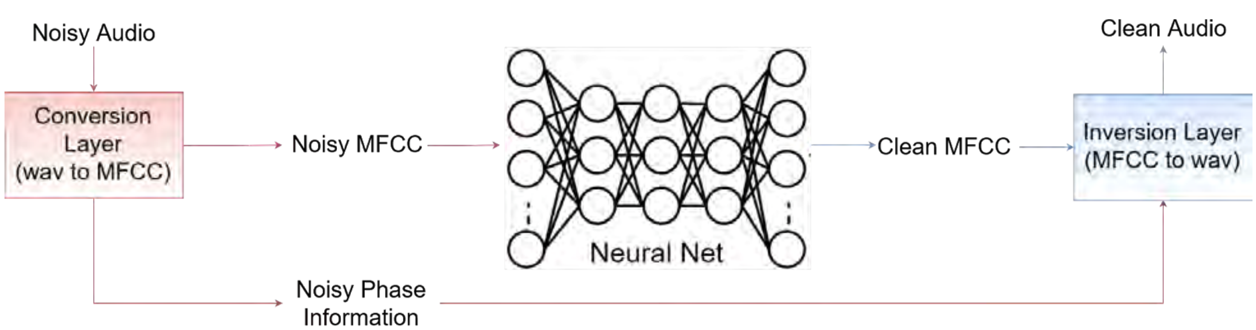


Figure 8, Software sequence overview to represent the flow of data through the program. Labels represent data states and all transformations.

3.5 CSCI Detailed Design

Below each CSCI is described and summarized at a high level of abstraction to discuss their purpose within the scope of the project. Further details on each CSCI can be found in the Appendix (Section 9.1) of this report. The appendix contains detailed descriptions of all functions, a usage manual, class/sequence diagrams for all software components, and commented code examples.

3.5.1 File I/O CSC

The File I/O CSC will allow the software to read in audio files of .wav format for further processing within our software. Additionally it will be responsible for the final providing the final output of our software by producing an audio file of .wav format that is a de-noised version of the input file sample. It must be written in “Python” in order to communicate well with the rest of our software and rely on the “Scipy” library. This CSC provides the main interface between the user and the software. The functions are defined as follows:

3.5.2 Wav Splitter/Splicer CSC

The Wav Splitter/Splicer CSC breaks down the original inputted wav files into roughly 5 second slices of the original files. This shall also take a series of 5 second .wav files and splice them together into a single .wav file. It shall be written in “Python” in order to communicate well with the rest of our software and it will rely on the “Scipy” library.

3.5.3 MFCC/Wav Converter/Inverter CSC

The MFCC/Wav Converter/Inverter CSC is designed to convert a given audio sample into a vector of MFCC coefficients for input into the main Autoencoder Neural Network. The converter is responsible for both to conversion of an audio sample to an MFCC vector as well as converting an MFCC vector into an audio sample. This CSC will be written in “Python” for compatibility with the rest of the software and will rely on the “Numpy” library for complex mathematical operations.

3.5.4 Neural Network Autoencoder CSC

The Autoencoder CSC is a neural network architecture that takes in noisy MFCC files and outputs cleaned MFCC files. It will first be trained using matched pairs of noisy and clean MFCC files and a variant of the Stochastic Gradient Descent algorithm. Once trained, the Autoencoder will act as a black-box that takes noisy MFCC files as input and outputs cleaned MFCC files. This CSC will be written in “Tensorflow” and can interact with the other “Python” subassemblies.

3.6 Requirements Traceability

The below table shows which requirements each CSC is responsible for, and how much they must accommodate specific requirements compared to other CSCs.

Note: This table is very similar to our System Requirements Flowdown from Section 3.6.2 because many of our CSCs overlap with our System Sub-assemblies.

System Requirement	CSC			
	File Input/Output	**Wav splitter/splicer**	**MFCC/Wav Converter**	**Neural Network Autoencoder**
1.1 Software Compatibility	1.0 Direct flow	1.0 Direct flow		
1.2 User Interface	2.0 Direct flow			
2.1 Platform	3.0 Direct flow			
2.2 Data Storage	4.0 Direct flow			
3.1 Memory Usage		2.0 Allocated, 100 Megabytes of RAM	1.0 Allocated, 100 Megabytes of RAM	1.0 Allocated, 1.8 Gigabytes of RAM
3.2 Software Delay		3.0 Allocated, 1 minute processing per input minute	2.0 Allocated, 1 minute of process per input minute	2.0 Allocated, 3 minutes processing per input minute
3.3 Software Size	5.0 Allocated, 5 MegaBytes	4.0 Allocated, 10 MegaBytes	3.0 Allocated, 10 Megabytes	3.0 Allocated, 75 Megabytes
3.4 Software Efficacy		5.0 Direct flow	5.0 Direct Flow	4.0 Direct flow

Table 3, Requirements Traceability

3.7 Notes

3.7.1 Software Usage

Our Software will not require a final GUI to interface with as it will be added to the existing Rescue 21 software in deployment

For testing a demonstration purposes, our software can be used to manually filter audio sample via linux command:

```
R21VoiceFilter  <input .wav path>  <output .wav path>
```

`<input .wav path>` = file path to the noisy audio file
`<output .wav path>` = file path for the cleaned audio file

3.7.2 Training and Testing Data

Because this project relies heavily on the data we use to train our Autoencoder, how we acquire our training data is of great importance. One option would be to use actual transmissions received by the Coast Guard, but even if we could procure that data, there would be no labeled set, no ground truth to have our model train on as a solution vector.

If a ground truth is needed, another option would be to generate noise such as Additive White Gaussian Noise, and add it to the clean audio. Though that would allow for functionally unlimited data, it would also lack many types of noise that are a result of natural distortion, like popping and amplitude clipping, and the model could overfit to those types of noise.

For our training data, we decided to take clean audio and distort it through natural means: transmitting the audio via radio waves in suboptimal conditions. Though the data collection would be slow, it would give a much more random collection of noise, and lead to greater generalization for our model. The data we have collected so far is a combination of transmissions through cell phones, VHF walkie-talkies, and HAM radios.

3.7.3 Appendix

Additional information regarding the software design is available in the Appendix. The appendix includes, class and activity diagrams, descriptions of all functions, snapshots of all code, and usage manuals. Relevant section is 9.1.

4.0 Acceptance Test Procedure

Acceptance Test Procedures are outlined in more detail in the Appendix (Section 9.2 through 9.8). In these sections are procedures for all test procedures, required equipment, as well as the data sheets used to conduct the tests for our sponsor.

4.1.1 Software Compatibility Test Procedure

This acceptance test ensured that the software could be deployed on all operating systems that Rescue 21 would run on. Since many details of Rescue 21 are classified, we estimated what operating systems may still be in use by the Coast Guard. Windows XP, Windows Vista, Windows 7, Windows 8, and Windows 10 were all tested for compatibility. To test this we ran the executable version of our software and made sure it output a cleaned audio file .

4.1.2 Memory Usage Test Procedure

This acceptance test ensured that the software could be run on a variety of systems with possibly a low allotment of RAM, and would be efficient enough to run on older, or less powerful computers. We estimated an upper bound of 2.0 GB. To test this we ran the executable version of our software and tracked it memory usage.

4.1.3 Processing Speed Test Procedure

This acceptance test ensured that the software would operate quickly, saving significant time over manual filtering methods. We estimated an upper bound of 5 minutes processing time per 1 minute of audio. To test this we ran the executable version of our software with a 1 minute audio file as an input, and recorded how long it took the program to execute.

4.1.4 Software Efficacy Test Procedure

This acceptance test ensured that the software would clean the audio effectively, allowing to be a viable substitute for a manual filter operator should the need for signal denoising be necessary. We estimated the Mean Squared Error value for processed audio would be below 56975. To test this we ran the executable version of our software and calculated the MSE of the processed file.

4.1.5 Appendix

Additional information regarding the test procedures is available in the Appendix. This includes in-depth test procedures as well as relevant data sheets. Relevant section is 9.2.

5.0 Models and Analyses

5.1 Memory Usage Model

The system requirements verification matrix requires analysis on memory usage. Reasonable memory consumption is essential for ensuring that the deep recurrent autoencoder and MFCC converter/inverter can run effectively on all types of systems.

Memory consumption can be monitored with a variety of tools, such as the process debugger or the top command. Baseline memory metrics will be estimated in Excel due to the large amount of time required to train and deploy a deep neural network.

There are several parameters to consider that directly affect the memory consumption of the deployed model, described in the table below. Especially notable is the wav sampling rate/MFCC slice size (a higher sampling rate over *t* seconds will consist of more data points than a lower sampling rate, which may strain the MFCC converter and inverter respectively) and the shape of the neural network (wider hidden layers and deeper neural networks are more computationally taxing than narrow hidden layers and shallow neural networks).

Requirement	Model	Tool	Inputs	Outputs
4.3.1: Memory Usage	Total Memory Consumption	Excel / Process Debugger (a command line tool)	• Input File Size • Wav Sampling Rate • Neural Network Depth	• RAM Usage

Table 4, Memory Usage Analysis

Our primary objective with respect to memory consumption is to create a system which consumes less than 2Gb of RAM upon deployment. This requires finding a reasonable balance between time slice and sampling rate, as well as neural network shape. MFCC coefficients are relatively low in dimensionality when contrasted with potential alternative architectures, such as fully convolutional autoencoders. Consequently, we believe that 1Gb of RAM upon deployment is a feasible prediction, as outlined below.

Requirement	Value	Prediction
4.3.1: Memory Consumption	Memory Consumption: <2Gb upon deployment	Memory Consumption: ~1Gb (assuming larger than expected network)

Table 5, Memory Consumption Prediction

Note that this excludes the training phase of the model, which will require much more computational power and is additionally dependent on the batch size used. As a result, the model should be trained *before* it is delivered to the R21 system on which it will be deployed.

Results:

Requirement	Model	Tool	Inputs	Prediction	Margin
4.3.1: Memory Usage	Total Memory Consumption	Excel / Process Debugger (a command line tool)	- Input File Size - Wav Sampling Rate - Neural Network Depth	~1.2GB (assuming larger than expected network)	0.8 GB

Table 6, Memory Usage Analysis Results

Sub-Assembly	Memory Usage (MiB)
Wave Splitter/Splicer	21.059 MiB
MFCC Converter/Inverter	5.582 MiB
Neural Network	134.879 MiB
Total:	373.602 MiB

Table 7, Sub-Assembly Memory Usage

6.0 Acceptance Test Results

6.1 System Requirements Final Acceptance Test Summary

The following provides the FAT results for all of the requirements. For signed test results as well as more in-depth results, consult sections 9.9 in the appendix.

Table 31: System Requirement Verification				
Requirement	**Method**	**Limit/Reference**	**Measured/Ref Value**	**Pass/Fail**
1. Interface Requirements	-	-	-	-
a. *Software Compatibility:* The software shall be compatible with Windows operating systems and be able to run on all necessary computers that currently run Rescue 21 software.	T, D	Compatible with Windows XP, Windows Vista, Windows 7, Windows 8, and Windows 10.	N/A	**Fail***
b. *User Interface:* The user shall not require anything more than a keyboard, mouse, and monitor in order to interact with this software.	D	Software can be used with only mouse, keyboard, and monitor	N/A	**Pass**
2. Environmental Requirements	-	-	-	**-**
3. Customer Constraints	-	-	-	**-**
a. *Platform:* Machine Learning training will take place on a cloud desktop hosted by AWS EC2.	I	All software training conducted on EC2 and verified through log files	N/A	**Pass**
b. *Data Storage:* File backups and archives will be stored on AWS S3.	I	All file backups can be found on S3 including: full .wav files, all file partitions, all code, and all MFCC vectors	N/A	**Pass**
4. Performance Requirements	-	-	-	-
a. *Memory Usage:* For any given input file, the finalized software shall not consume more than 2 Gigabytes of RAM when deployed to the host computer.	T, A	2 GB	373.602 MB	**Pass**
b. *Software Delay:* An input file of duration 1 minute at 48 kHz resolution and 16 bits/sample shall take no longer than 5 minutes for the finalized software to progress from start to finish by analyzing and producing a corresponding output file.	T	5 Minutes	8.031 Seconds	**Pass**

c. *Software Size:* The finalized software shall not consume more than 100 Megabytes of disk storage space when installed on the host computer.	I	100 MB	70.5 MB	**Pass**
d. *Software Efficacy:* The software shall produce output audio files that are more intelligible than noisy input audio files.	T, D	$2\ \|\|\hat{y} - y\|\|^2 < \|\|x - y\|\|^2$ $\hat{y}$: output MFCC vector y: original target MFCC vector X: input noisy MFCC	N/A	**Fail**

Table 8, System Verification Results Table

6.2 Appendix

Additional information regarding the test results is available in the Appendix. This includes all results for all sub-assemblies. Relevant section is 9.9.

7.0 Final Budget

Below is a table that describes our **final** cost summary. Our total budget was under the expected amount due to AWS resources not factoring in to our final budget.

A large part of the budget was allocated in the end towards audio equipment for final analysis of the processed audio. It was advised by a mentor that the equipment at least similar in quality to equipment used in audio labs.

Our budget also included accommodations for all design day materials including shirts and the poster / cost of printing our final poster.

Item	Price	Estimated Total	Spent	Description
Audio Analysis Equipment	$1366.95	$1700	$1366.95	High quality monitors, amplifiers and digital audio converters for final analysis.
Demonstration Audio Equipment	$649.77	$700	$649.77	Closed, circumaural headphones for design day demonstration
External Solid State Hard Drive	$319.98	$319.98	$319.98	External SSD for quick local transfer of 50+ GB of audio data in various forms.
Design Day Shirts	$500	$500	$335	Team shirts
Design Day Poster	$150	$150	$85.86	Poster
Midland GXT1000	$75.33	$75.33	$75.33	VHF “Walkie-Talkies” needed to collect additional data
Total Budget	**-**	**$3940.75**	**$2832.89**	**$1167.11 Left**

Table 9, Final Budget

8.0 Lessons Learned

Machine Learning

Through this project, we have learned how to effectively gather large amounts of training data necessary for training our model. One of the most important lessons learned regarding this is that spending more time planning out how to collect data earlier will increase efficiency later.

We have also learned a great deal about Machine Learning as well as its nuances that can make it difficult to implement.

Audio/Signal processing

This project required a thorough understanding of signal processing to understand how voice enhancement can be done via software. As a result we had to learn how audio is represented digitally, how to convert it to a more compressed form (Mel Frequency Cepstral Coefficients), as well as how to restore audio from the compressed form and why this might cause quality loss. This required a thorough literature survey on available methods for MFCC inversion and estimators for signals with phase deficits.

Project Approach

With respect to this specific project, we learned that not all projects work best under a “waterfall” approach. In essence, this project was mainly one of research and development and thus required lots of constant reworking, reorganization, and tons of testing of different methods. Under the waterfall approach, the entire project design is roadmapped from the beginning. What this project required was the testing and validation of many different methods to see which methods produced the best results.

Communication

We learned that constant communication between group members as well as with the sponsor was crucial for project success. Everyone needed to be on the same page so as to meet customer and course deadlines. In addition, communication with our sponsor was important to ensure that there were no surprises in our progress and that they were aware of any and all roadblocks that we may have hit.

9.0 Appendix

9.1 Software Design Document

Advanced Voice Filtering for R21 using Machine Learning
Software Design Document

Document Number 100,100

Team 17053
Sam Gianelli, Patrick Wegner, Alexander Brooks, Jodi Rickel, Michael Ragone

January 23 2018 rev (-)
Prepared for General Dynamics

Table of Contents

9.1.0 CSCI Detailed Design

The unit design decisions, constraints, language, procedural commands, data handling, logic, and algorithms are provided for each CSC below:

Note: All sub headers reference the original header that they were summarized in earlier in this report.

3.5.1 File I/O CSC

The File I/O CSC will allow the software to read in audio files of .wav format for further processing within our software. Additionally it will be responsible for the final providing the final output of our software by producing an audio file of .wav format that is a de-noised version of the input file sample. It must be written in "Python" in order to communicate well with the rest of our software and rely on the "Scipy" library. This CSC provides the main interface between the user and the software. The functions are defined as follows:

Function	**Input**	**Output**	**Description**
inputFile	Path to dir of .wavs	The user designated .wav file(s)	Takes a user specified file/directory of files into the software
outputFile	Path to output dir of .wavs	The processed .wav file(s)	Creates an output file/directory containing cleaned audio files.

Table 10, File/IO CSC Functions

3.5.2 Wav Splitter/Splicer CSC

The Wav Splitter/Splicer CSC breaks down the original inputted wav files into roughly 5 second slices of the original files. This shall also take a series of 5 second .wav files and splice them together into a single .wav file. It shall be written in “Python” in order to communicate well with the rest of our software and it will rely on the “Scipy” library.

Function	**Input**	**Output**	**Description**
splitFile	Single .wav file	Fractionally overlapping .wav files of length ~5 seconds	Splits an input .wav file into many ~ 5 second .wav files such that the union of all files covers the original file and adjacent .wav files fractionally overlap
spliceFile	Fractionally overlapping .wav files of length ~5 seconds	Single .wav file	Splices the overlapping ~5 second .wav files into a single output .wav file

Table 11, Wave Splitter/Splicer CSC Functions

The programmatic steps for splitting a file are outlined below:

splitFile():

1. Assign variable
    a. Wav_file = argv[1] (path passed as a command line argument)
2. Load wav file using Scipy.io
3. Extract data from Scipy.io output
    a. Define samp_rate = wav.open(Wav_file)   (sample rate)
    b. Define wav_duration = wav.open(Wav_file)   (time duration of .wav file)
4. Use the sample rate to compute:
    a. Slice length = wav_duration / samp_rate
    b. Split point to separate wav files
5. Loop over split points and write each wav to output directory
    a. For i in split points
        i. Write(.wav) to “output directory/Wav_file[i].wav”

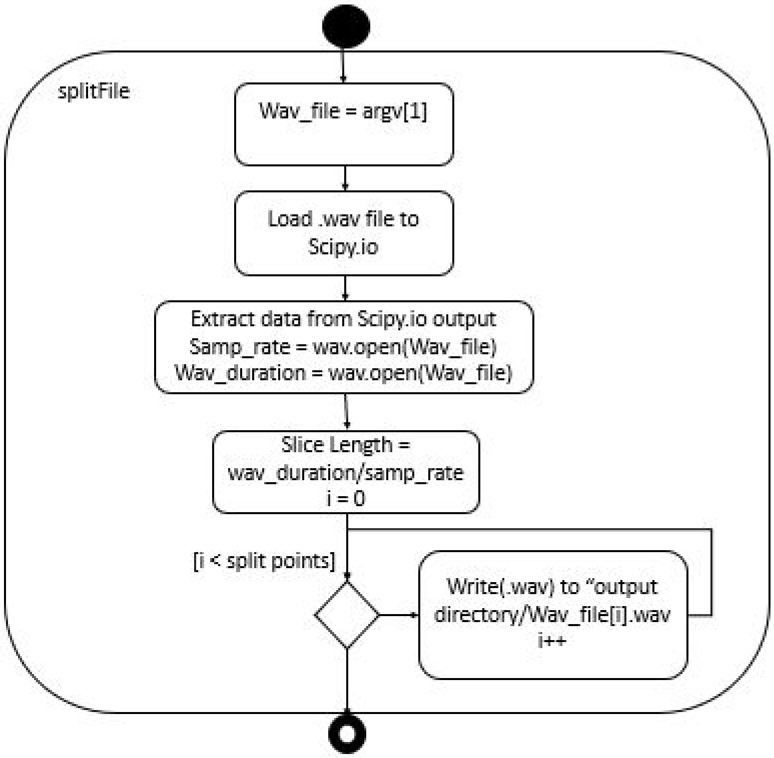


Figure 9, Activity Diagram for splitFile

The programmatic steps for splicing a file are outlined below:

spliceFile():

1. Assign vector of variable
    a. Wav_files = argv[n] (path base name and number passed as command line argument)
2. Load ordered vector of wav files to be spliced from wav location directory using Scipy.io
3. Extract data from Scipy.io output
    a. Define samp_rate = wav.open(Wav_file)   (sample rate)
4. Loop over n files in Wav_files and concatenate
    a. Output = []
    b. For i <= n

    i. Output = Output + Wav_files[n]

5. Write(Output) to “output_directory/Output.wav”

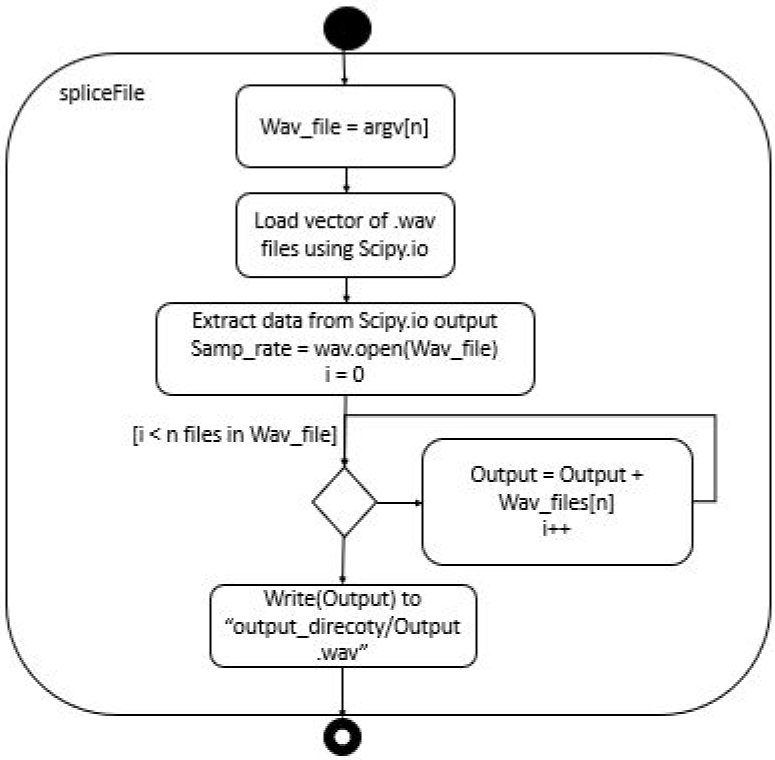


Figure 10, Activity Diagram for spliceFile

3.5.3 MFCC/Wav Converter CSC

The MFCC/Wav Converter CSC is designed to convert a given audio sample into a vector of MFCC coefficients for input into the main Autoencoder Neural Network. The converter is responsible for both to conversion of an audio sample to an MFCC vector as well as converting an MFCC vector into an audio sample. This CSC will be written in "Python" for compatibility with the rest of the software and will rely on the "Numpy" library for complex mathematical operations.

Function	Input	Output	Description
wav2MFCC	.wav file	MFCCs	Converts .wav file into a file of MFCCs with standard equation
MFCC2wav	MFCC vector	.wav file	Converts a file of MFCCs into a .wav with standard equation
load_wavs	Directory path	.wav file vector	Load wav files from a directory
frame	Signal vector	Signal vector	Frames the signal into a list of overlapping slices
framify_waves	Signal vector, sampling rate, frame duration, frame stride	Signal vector	Slices wav signal vector into separate frames
save_waveforms	Signal vector	Void, shows waveform visualization	If debug is enabled, plots a random waveform (for debugging pipeline)
signal_reconstruction	MFCC vector	.wav file	Converts all mfccs to wav representation, then uses the wav_splicer module to convert everything back together as a completed sample

Table 12, MFCC/Wav Converter CSC Function

The programmatic steps for converting a wav file to MFCC are outlined below:

wav2MFCC(Wav_file):

1. Load wav file
    a. Wav_file = argv[1] (path passed as a command line argument)
    b. Wav = wav.open(Wav_file)
2. Convert Stereo to Mono via averaging two channels
3. Frame the signal (same as file splitter with overlaps between slices)
4. Apply Hamming window function, i.e. w(n) = 0.54 - .46cos(2*pi*n / (N - 1)), to each frame.
5. Evaluate spectra, i.e. (spectra = np.fft.rfft(frame_wav, N))/N), then split into power spectrum aka periodogram and phase spectrum:
    a. power_spectra = spectra.^2
    b. phase_spectra = angle(spectra). Save to file.
6. Apply mel filterbank equations (spectrogram) to power_spectra.
7. Apply discrete cosine transform (MFCC)

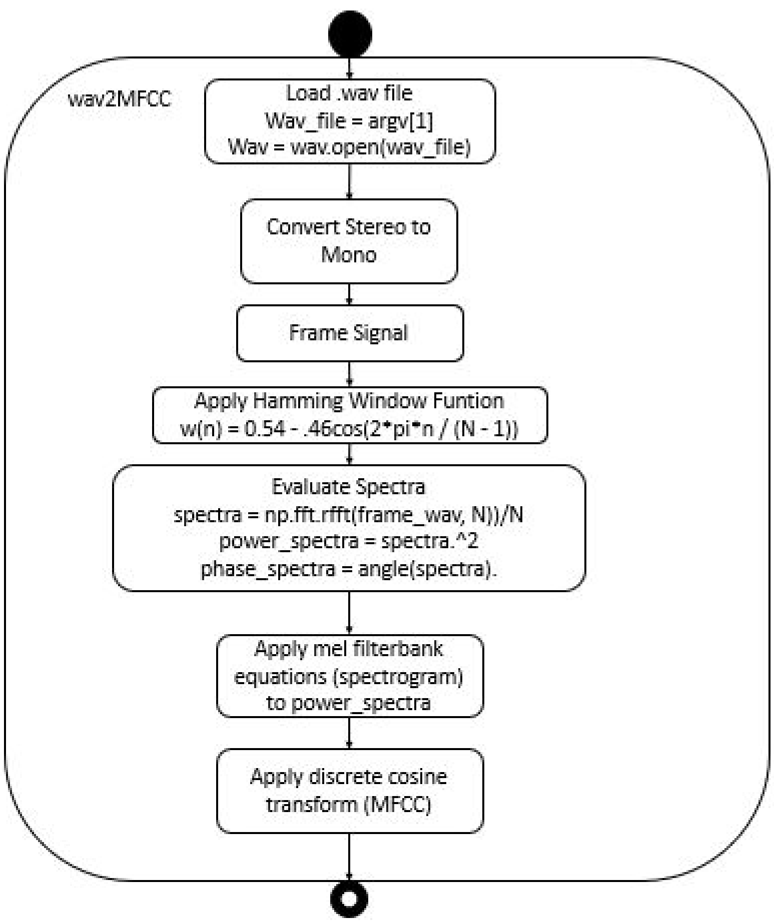


Figure 11, Activity Diagram for wav2MFCC

The steps for converting a MFCC file to a wav file are similar, but in reverse order. Note that phase inversion is highly nontrivial, so we choose to save noisy phase and apply it to the output power spectrum.

1. Load MFCC, apply inverse discrete cosine transform.
2. Apply Moore-Penrose pseudoinverse of mel filterbank equations.
3. Apply phase_spectra to power_spectra reconstruct full spectrum
    a. Spectra = power_spectra * e^(j*noisy_phase_spectra)
4. Apply inverse hamming window to each frame.
5. Splice frames to reconstruct signal.
6. Save .wav file.

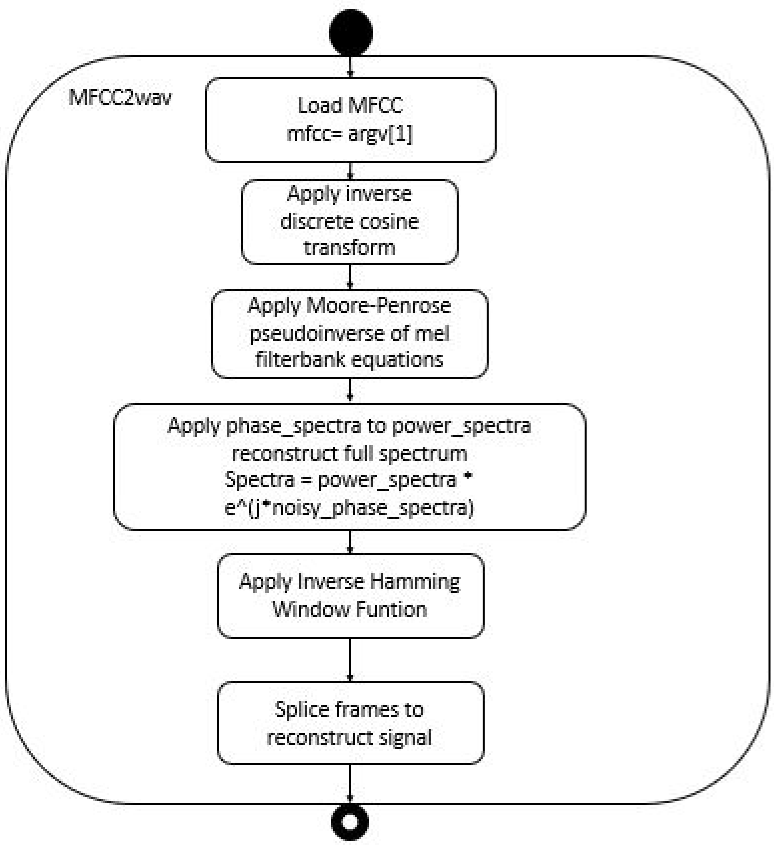


Figure 12, Activity Diagram for MFCC2wav

3.5.4 Neural Network Autoencoder CSC

The Autoencoder CSC is a neural network architecture that takes in noisy MFCC files and outputs cleaned MFCC files. It will first be trained using matched pairs of noisy and clean MFCC files and a variant of the Stochastic Gradient Descent algorithm. Once trained, the Autoencoder will act as a black-box that takes noisy MFCC files as input and outputs cleaned MFCC files. This CSC will be written in "Tensorflow" and can interact with the other "Python" subassemblies.

Function	Input	Output	Description
main	MFCC Directory	N/A	Takes an input noisy MFCC file and produces a directory of cleaned output MFCC files
train_model	Train_set, Test_set, Num_epochs, Batch_size	Model accuracy	Takes an input noisy MFCC file and passes it to the trained autoencoder. Runs optimizer (SGD) and cost minimization on the computation graph in a Tensorflow session for a preset number of epochs, displaying model accuracy as training progresses
define _model	Data (input to first layer in neural network)	Output (final layer in neural network)	Defines the neural network structure. In Tensorflow, this is the construction of the computational graph, which can then be instantiated in a session
partition_data	Ratio	Test_set, Train_set	Takes in a percentage, which is used to randomly split the MFCC data into training and testing sets. Returns list of tuples corresponding to file name and label name for each set. Note that in this case, each label is also an MFCC (autoencoder)
load _MFCCs	Filepaths	MFCC	Loads MFCCs as a matrix of size n x m, where n is the batch size (length of the Filepaths list) and m is the number of coefficients used

Table 13, Neural Network Autoencoder CSC Function

Primary Class: DeepNet

DeepNet
model_name sess saver activation_function activation optimizer_function optimizer loss_function cost learning_rate batch_size num_epochs epoch_step momentum model_architecture h1_neurons h2_neurons h3_neurons response_size layer_1_res layer_2_res layer_3_res weights biases input_placeholder output_placeholder output model_path save load recurrence_length validate train
__init__(self, model_name,activation_function,optimizer_function, loss_function,learning_rate, batch_size, num_epochs, epoch_step, momentum, model_architecture, num_neurons, data_size, model_path, save, load) generate_placeholders(self) generate_model(self) define_architecture(self) construct_model(self, data_size) create_cost_node(self) create_optimization_node(self) model_initializer(self, load) fit_model(self, training_noisy, training_clean, testing_noisy, testing_clean) train_model(self, training_noisy, training_clean, testing_noisy, testing_clean) train_step_NN(self, noisy_data, clean_data) train_step_RNN(self, noisy_data, clean_data) validate_step_NN(self, noisy_data, clean_data) validate_step_RNN(self, noisy_data, clean_data) generate_batch_inference(self, data) generate_single_inference(self, data) load_model(self, model_path, data_size) print_debug(self)

Figure 13,Class Diagram for DeepNet

Class Members:

Note: **Bolded** member values indicate values that can be specified/overwritten by the user upon instantiation of the model

Member and Default Value	Descriptions
self.model_name = 'R21_DDAE' self.sess = None self.saver = None	- Name of the model - Tensorflow session of the model, necessary for running/saving/restoring the model - A Saver object, needs instantiation to Save/Restore

self.activation_function = 'sigmoid'
 	
self.activation = None
self.optimizer_function = 'adam_optimizer'
 	
self.optimizer = None
self.loss_function = 'squared_error'
	- Specifies which activation function to use. Available functions are:
	    - sigmoid, tanh, relu, none
	- Function handle for selected activation function
	- Specifies which optimizer function to use. Available functions are:
	    - Stochastic gradient descent, Adam, Momentum, rms_prop
	- Function handle for selected optimizer function
	- Specifies which loss function to use. Available functions are:
	    - Squared_error, soft_max, mean_squared

self.cost = None **self.learning_rate** = 0.00005 **self.batch_size** = 1000 **self.num_epochs** = 30 **self.epoch_step** = 1 **self.momentum** = 0.5	- Function handle for selected loss function - Neural Network learning rate (manually tuned) - Number of samples to train on before applying backpropagation to model - Number of iterations to train the model - Number of epochs to run before running validation set - Momentum parameter for momentum optimizer

self.model_architecture = 'DDAE'	- Specifies which Neural Network model to use:
	    - DDAE : Deep Denoising Autoencoder
	    - DDRAE : Deep Denoising Recurrent Autoencoder

self.h1_neurons = 500 **self.h2_neurons** = 500 **self.h3_neurons** = 500 **self.response_size** = 40	- Number of neurons within hidden layer 1 - Number of neurons within hidden layer 2 - Number of neurons within hidden layer 3 - Size of the input and output layers (equal)
self.layer_1_res = None self.layer_2_res = None self.layer_3_res = None	- TF variable holding layer 1 operations - TF variable holding layer 2 operations - TF variable holding layer 3 operations
self.weights = None	- TF variable holding weights for all model layers

self.biases = None self.input_placeholder = None self.output_placeholder = None self.output = None **self.model_path** = 'model-99.ckpt' self.save = False self.load = False **self.recurrence_length** = 3 self.validate = self.validate_step_RNN self.train = self.train_step_RNN	- TF variable holding bias values for all layers - TF placeholder for the input to the model - TF placeholder for the output of the model - TF variable holding results of final model operations - Path the save the model after training - Boolean to toggle saving - Boolean to toggle loading - Size of the recurrence window (only for DDRAE) - Function handle for validation step when using DDRAE - Function handle for validation step when using DDRAE

Table 14, Member Values for DeepNet

Class Methods:

def __init__(self, model_name='R21_NN', activation_function='sigmoid',
optimizer_function='adam_optimizer', loss_function='squared_error',
learning_rate=0.00005, batch_size=64, num_epochs=30, epoch_step=1,
momentum=0.5, model_architecture='DDAE', num_neurons=500, data_size=40,
model_path='tmp/model-99.ckpt', save=False, load=False):

Constructor for the DeepNet class. See above table for descriptions of members.

def generate_placeholders(self):

Generates the self.input_placeholder and self.output_placeholder member values based on the preset self.response_size member.

def generate_model(self):

Generates the self.weights and self.biases member values based on the following preset members:

self.h1_nuerons, self.h2_neurons, self.h3_neurons, self.response_size.

def define_architecture(self):

Generates the operations conducted at each layer of the neural network. This includes self.layer_1_res, self.layer_2_res, self.layer_3_res, and self.output class members. This method also defines which activation function is used on each layer of the network.

Options for Model Architectures:

- Deep Denoising Autoencoder (DDAE)
- Deep Denoising Recurrent Autoencoder (DDRAE)

Options for Activation Functions:

- Sigmoid
- Relu
- Tanh
- None

PseudoCode:

- If self.model_architecture == ‘DDAE’:
    - Call **def DDAE_model**(self)
- If self.model_architecture == ‘DDRAE’:
    - Call **def DDRAE_model**(self)

def construct_model(self, data_size):
Wrapper method that calls all class methods necessary to completely instantiate the Tensorflow model such that it is ready to be trained or used to generate inferences.

Instantiation Procedure:
- **generate_placeholders()**
- **generate_model()**
- **define_architecture()**
- **create_cost_node()**
- **create_optimization_node()**

def create_cost_node(self):
Generates a function handle in self.cost that executes the cost function during the minimization of the model.
Options include:
- Squared_error
- Soft_max
- mean_squared

def create_optimization_node(self):
Generates a function handle in self.optimizer that executes the optimization function on the model cost during the minimization of the model.
Options include:
- Stochastic gradient descent
- Adam optimizer
- Momentum optimizer
- Rms propagation

def model_initializer(self, load=False):
Initializer script to initialize all global and local variables declared within the model for its construction.

def fit_model(self, training_noisy, training_clean, testing_noisy, testing_clean):
Wrapper function for constructing a model, starting a session, and training. Accepts training/testing data and training/testing labels. Once training is completed, the model will be saved to the model path attached to the class instance.

def train_model(self, training_noisy, training_clean, testing_noisy, testing_clean):
Iterates through a pre-specified number of epochs and trains with batched data. Also prints the test set error as epochs progress.

def train_step_NN(self, noisy_data, clean_data):
Trains the non-recurrent model for a single epoch.

def train_step_RNN(self, noisy_data, clean_data):
Trains the recurrent model for a single epoch.

def validate_step_NN(self, noisy_data, clean_data):
Tests the non-recurrent model (forward step only).

def validate_step_RNN(self, noisy_data, clean_data):
Tests the recurrent model (forward step only).

def generate_batch_inference(self, data):
Main deployment method after training has finished. Accepts a a matrix of data to generate inferences upon. Return a matrix of inferences that can be reassembled into a wav file.

def generate_single_inference(self, data):
One dimensional analog of generate_batch_inference().

def load_model(self, model_path=None, data_size=40):
Checks model_path for a valid model to load from, creates self.saver, constructs the model, then loads the values from the saved model file to the newly constructed model.

def print_debug(self):
Helper method to print out ALL class members of the particular object instantiation of DeepNet. Not necessary for deployment, but helpful if errors occur.

List of command line arguments and flags:

- --analyze
  - Allows debug mode wherein accuracy scores are recorded and debugging/visualization data is saved in an image.

The programmatic steps for *creating/training* the autoencoder architecture are outlined below:

1. Assigned variables
   a. Batch_size = 32(for training the neural net)
   b. Layer_sizes = [500 per layer] (number of neurons in each layer of the autoencoder)
   c. Learning_rate = (for controlling how fast the neural net learns)
   d. Optimizer = Stochastic Gradient Descent
   e. Noisy_MFCC_dir = '~/ … ' (input)
   f. Clean_MFCC_dir = '~/ … ' (labels)
   g. test_ratio = 0.1 (percentage of samples assigned to test set; the rest are assigned to train set)
2. Create model (Tensorflow computation graph)
3. Load MFCC directories (input and labels)
4. For each pair in MFCC data
   a. Generate a random number between 0 and 1.
   b. If random < test_ratio, assign pair to test_set. Else assign to train_set.
5. Instantiate model
6. Assign optimizer (Momentum) and cost function
7. Train model (with shuffling active) for a preset number of epochs. For each batch:
   a. Load MFCC representation of training set batch (randomly selected)
   b. Run session over optimizer and cost on the defined model
   c. Evaluate accuracy on test MFCCs
   d. Record epoch loss
8. If --analyze is enabled, save plot of loss as a function of epochs to current working directory
9. Save neural network state

The programmatic steps for *deploying* the autoencoder architecture are outlined below:

1. Load noisy Wav file
2. Convert noisy Wav to MFCCs
3. Restore the deep neural network (autoencoder)
4. Forward step MFCC through the autoencoder
5. Convert the resulting MFCCs to Wav
6. Display approximate loss
   a. Loss = distance the results are from the ground truth
7. save wav to the current working directory

9.1.5 Supplementary Software Notes

Software Usage

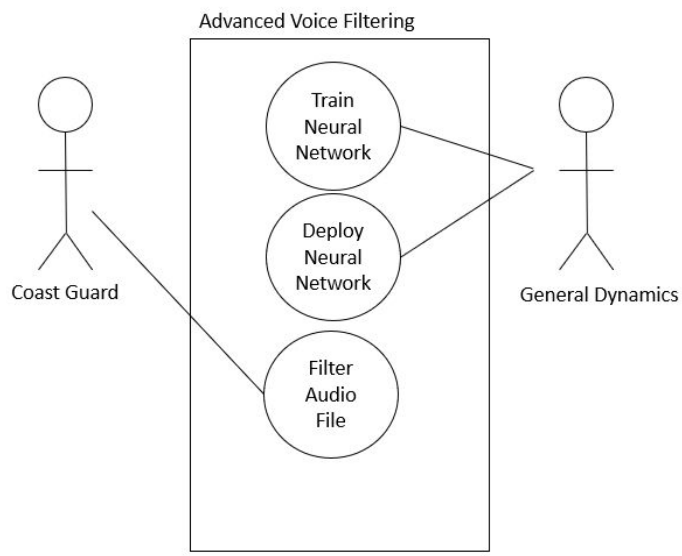


Figure 14, Use Case Diagram

Our Software will not require a final GUI to interface with as it will be added to the existing Rescue 21 software in deployment

For testing a demonstration purposes, our software can be used to manually filter audio sample via linux command:

```
R21VoiceFilter  <input .wav path>  <output .wav path>
```

`<input .wav path>` = file path to the noisy audio file

`<output .wav path>` = file path for the cleaned audio file

Training and Testing Data

Because this project relies heavily on the data we use to train our Recurrent Autoencoder, how we acquire our training data is of great importance. One option would be to use actual transmissions received by the Coast Guard, but even if we could procure that data, there would be no labeled set, no ground truth to have our model train on as a solution vector.

If a ground truth is needed, another option would be to generate noise such as Additive White Gaussian Noise, and add it to the clean audio. Though that would allow for functionally unlimited data, it would also lack many types of noise that are a result of natural distortion, like popping and amplitude clipping, and the model could overfit to those types of noise.

For our training data, we decided to take clean audio and distort it through natural means: transmitting the audio via radio waves in suboptimal conditions. Though the data collection would be slow, it would give a much more random collection of noise, and lead to greater generalization for our model. The data we have collected so far is a combination of transmissions through cell phones, VHF walkie-talkies, and HAM radios.

Steps taken to generate and use the training and testing data are outlined in the following section:

Training and Deploying the Neural Network

In the event that our sponsors want to retrain our neural network, the following steps outline such a procedure:

Deployment:

To deploy the Neural Network from a pretrained model, all that is necessary is to execute the RunDeepNet executable:

```
>> python RunDeepNet -i <input filename> -o <outputfilename>
    -i: Input Filename, must be a .wav file
    -o: Output Filename, must be a .wav file
    -c: (optional) Disables the neural network, simply
        converts the input file to MFCCs and back to .wav
```

To deploy the Neural Network from our library methods, see the code for the main() method of the RunDeepNet.py script in the Appendix (9.XX). The necessary steps summarized are:

1. Use `wave_splitter.split_wav(...)` to split your input file into smaller file fragments.
2. Use `mfcc.wav_2_mfcc(...)` to convert all file fragments into MFCC vectors
3. Use `DeepNet.load_model(...)` to load the neural network model of choice.
4. Use `DeepNet.generate_single_inference(...)` to create a new denoised MFCC matrix
5. Chain `mfcc.mfcc_2_spectra(...)` and `mfcc.spectra_2_wav(...)` to recreate the audio file from the new MFCC matrix.

Training:

In order to train the Neural Network, you must first have valid data to train it with. Data is expected to be stored as a 4-dimensional .npy matrix. The dimensions are described as:

```
    (M 5-second audio samples, N MFCC coefficients, 332 frame overlaps,
clean/noisy)
```

Where the last clean/noisy dimension is a toggle between the coefficients corresponding to the clean audio sample and the noisy equivalent for the same sample. Clean = 0, Noisy = 1.
For example, in a dataset consisting of 40 MFCC coefficients, the expected dimensions are:

```
    (M, 40, 332, 2)
```

Where M depends on the length of the input audio.

If you already have this .npy matrix, skip this paragraph. To generate this dataset, follow

1. Generate Training and Validation datasets.
   a. For this the use can either use our provided data set (accessible via Data_Utils.py) or use their own data.

2. Create a python script to instantiate and train the new network. This can be done in two lines of code:
    a. newNetwork = DeepNet(optional_args=)
    b. newNetwork.fit(w, x, y, z)
3. After running the fit method, the weights of the trained graph with be saved to a directory specified in the DeepNet constructor.
4. The saved computational graph can be accessed via:
    a. loadedNetwork = DeepNet(optional_args=)
    b. loadedNetwork.load_model(path)
5. Subsequent prediction can be made via the following methods:
    a. loadedNetwork.generate_single_inference(x)
    b. loadedNetwork.generate_batch_inference(x)

9.2 Acceptance Test Procedure

Advanced Voice Filtering for R21 using Machine Learning
Acceptance Test Procedure

Document Number 100,300

Team 17053
Sam Gianelli, Patrick Wegner, Alexander Brooks, Jodi Rickel, Michael Ragone

January 23 2018 rev (-)
Prepared for General Dynamics

9.2.0 Record of Changes

Rev	Date	Title of Changes	Description of Change
(-)	1/18/18	Initial Release	N/A

Table 15, Record of Changes

9.3 Table of Contents

9.4.1 Software Compatibility Test

9.4.2 Introduction: This procedure outlines the acceptance tests to be performed on the RNN program. Specifically this test verifies the standalone programs compatibility with various operating systems. Ideally the module should be compatible with Windows operating systems dating back to Windows XP. The workings of R21 are classified, so it is not known which operating systems

9.4.3 Referenced Documents: Advanced Machine Learning with R21 Recurrent Neural Network System Requirements Document, 10/5/2017, rev (-)

9.4.4 Required test equipment:

Description	Model Number	Accuracy
Computer/VM Running Windows XP OS	N/A	N/A
Computer/VM Running Windows Vista OS	N/A	N/A
Computer/VM Running Windows 7 OS	N/A	N/A
Computer/VM Running Windows 8 OS	N/A	N/A
Computer/VM Running Windows 10 OS	N/A	N/A

Table 16, Software Compatibility Required Test Equipment

9.4.5 Table of Tests:

Test #	Test	Requirement
9.4.1	Software Compatibility	Able to process file

Table 17, Software Compatibility Table of Tests

9.4.6 Step-by-step Procedure:

1. Install software onto target machine (hardware or virtual)
2. Put test files into appropriate directories
3. Attempt to run executable
4. Attempt to input a file
5. Run the program to process the file
6. Ensure that output is in correct location and is satisfactory
7. If a failure occurred at stages 3-6, record at what stages it failed, what operating system was used, and any additional details.

9.4.7 Support Requirements: N/A

9.5.1 Memory Usage Test Procedure

9.5.2 Introduction: This procedure outlines the acceptance tests to be performed on the Machine Learning Top Assembly (Document 101,000) to ensure proper memory usage (consumes less than 2 gigabytes of RAM when deployed on the host computer).

9.5.3 Referenced Documents: Advanced Machine Learning with R21 Recurrent Neural Network System Requirements Document, 10/5/2017, rev (-)

9.5.4 Required test equipment:

Description	Model Number	Accuracy
Computer Running Windows Operating System with TensorFlow installed	N/A	N/A

Table 18, Memory Usage Test Required Test Equipment

9.5.5 Table of Tests:

Test #	Test	Requirement
9.5.1	Deployment Test	Less than 2 Gb of RAM usage

Table 19, Memory Usage Test Table of Tests

9.5.6 Step-by-step Procedure:

1. Configure system. Move test data into same directory as neural net & wav splitter/splicer
2. Split the audio into several smaller wav files using the splitter
3. Convert small .wavs into MFCC form
4. Deploy the neural network in test mode (forward step only) on test data
5. Monitor memory usage via Top

9.5.7 Support Requirements: Computer must be able to run the ‘top’ command. Test should be performed on an R21 compatible system with identical processing power and memory capacity as the actual system.

9.6.1 Processing Speed Test

9.6.2 Introduction: This procedure outlines the acceptance tests to be performed on the Advanced Voice Filtering for Rescue 21 using Machine Learning System (Document 101,000). This test verifies that the System takes 5 minutes to process 1 minute of input of a .wav audio file at 48 kHz on a standard personal computer.

9.6.3 Referenced Documents: Advanced Machine Learning with R21 Recurrent Neural Network System Requirements Document, 10/5/2017, rev (-)

9.6.4 Required test equipment:

Description	Model Number	Accuracy
Computer Running Windows Operating System	N/A	N/A

Table 20, Processing Speed Test Required Test Equipment

9.6.5 Table of Tests:

Test #	Test	Requirement
4.6.1	Processing Speed	5 minute processing per minute input

Table 21, Processing Speed Table of Tests

9.6.6 Step-by-step Procedure:

1. Install software onto target computer
2. Acquire a 1 minute long 48 kHz .wav audio file with distorted voices. Choose distortion caused by communication channel noise
3. Start timer then start program.
4. Stop timer when program finishes. Record time “t” on data sheet. Verify that this time is less than 5 minutes.

9.6.7 Support Requirements: Tests should be performed on a compute running only necessary operating system software.

9.7.1 Software Efficacy Test

9.7.2 Introduction: This procedure outlines the acceptance tests to be performed on the Advanced Voice Filtering for Rescue 21 using Machine Learning System (Document 101,000). This test verifies that the System properly filters the audio. This is done by generating the MFCC vectors of the clean (y), dirty(x), and de-noised($\hat{y}$) audio and ensuring that $2\,||\hat{y} - y||^2 < ||x - y||^2$, where || represents the Euclidean Norm.

9.7.3 Referenced Documents: Advanced Machine Learning with R21 Recurrent Neural Network System Requirements Document, 10/5/2017, rev (-)

9.7.4 Required test equipment:

Description	Model Number	Accuracy
Computer Running Windows Operating System	N/A	N/A

Table 22, Software Efficacy Required Test Equipment

9.7.5 Table of Tests:

Test #	Test	Requirement
9.7.1	Software Efficacy	$2\,||\hat{y} - y||^2 < ||x - y||^2$

Table 23, Software Efficacy Table of Tests

9.7.6 Step-by-step Procedure:

1. For a sample of 10 “test” audio samples, generate MFCC vectors of the clean “ground truth” audio. (vector y)
2. Generate MFCC vectors of the distorted counterparts of each “test” audio sample (vector x)
3. Generate MFCC vectors of the audio samples output by our software, the de-noised samples (vector $\hat{y}$)
4. Calculate MSE (Mean-Squared Error) between y and $\hat{y}$: $||\hat{y} - y||^2$
5. Calculate MSE between y and x:
   $||x - y||^2$
6. Verify $2\,||\hat{y} - y||^2 < ||x - y||^2$ for all test samples.

9.7.7 Support Requirements: Tests should be performed on a compute running only necessary operating system software.

9.8 Data Sheet Templates

<table>
<tr><th colspan="4">Advanced Voice Filtering for Rescue 21 using Machine Learning Acceptance Test Data Sheet</th></tr>
<tr><td colspan="4">Referenced ATP Paragraph Number: 9.4.1</td></tr>
<tr><td colspan="4">Analysis Referenced (for verification by T/A): N/A</td></tr>
<tr><td colspan="4">Name of Test: Software Compatibility</td></tr>
<tr><td colspan="4">Unit Under Test (UUT):
Name: Executable Standalone Software
Part Number: n/a
Serial Number: n/a</td></tr>
<tr><td colspan="2">Results (Pass/Fail):</td><td colspan="2">Date of Test:</td></tr>
<tr><td>Operating System:</td><td>Virtual or Hardware:</td><td>Where Program Failure Occured:</td><td>Additional Notes:</td></tr>
<tr><td colspan="4">Analyses Results, if any:</td></tr>
<tr><td colspan="4">Signatures:
Tester ________________________________

Customer _____________________________</td></tr>
</table>

Advanced Voice Filtering for Rescue 21 using Machine Learning Acceptance Test Data Sheet

Referenced ATP Paragraph Number: 9.5.1

Analysis Referenced (for verification by T/A): Memory Usage Model 100,210

Name of Test: Memory Usage Test Procedure

Unit Under Test (UUT):
Name: Machine Learning Top Assembly
Part Number: n/a
Serial Number: n/a

Results (Pass/Fail):	**Date of Test:**

Recording of Test Measurement:	**Requirement (SRD, with Tolerances):**	**Test Equipment Error:**	**Adjusted Test LImit:**
Memory Usage, M =	< 2.0 Gb	.2 Gb (mitigate top fluctuation by averaging)	< 2.1 Gb

Computations, (Include Analyses Results, if any):

Record process memory usage from Top over 100 forward steps (m as a function of time) as well as time elapsed.

Calculate average memory usage: M = □m/t

Signatures:
Tester ________________________________

Customer _____________________________

Advanced Voice Filtering for Rescue 21 using Machine Learning Acceptance Test Data Sheet			
Referenced ATP Paragraph Number: 9.6.1			
Analysis Referenced (for verification by T/A): N/A			
Name of Test: Processing Speed Test			
Unit Under Test (UUT): **Name:**Advanced Voice Filtering for R21 using Machine Learning System **Part Number:** n/a **Serial Number:** n/a			
Results (Pass/Fail):		**Date of Test:**	
Recording of Test Measurement: Time to finish t =	**Requirement (SRD, with Tolerances):** < 5 minutes	**Test Equipment Error:** 1 second	**Adjusted Test LImit:** 1 hour
Computations, (Include Analyses Results, if any):			
Signatures: **Tester** ________________________________ **Customer** _____________________________			

<table>
<tr><th colspan="4">Advanced Voice Filtering for Rescue 21 using Machine Learning Acceptance Test Data Sheet</th></tr>
<tr><td colspan="4">Referenced ATP Paragraph Number: 9.7.1</td></tr>
<tr><td colspan="4">Analysis Referenced (for verification by T/A): N/A</td></tr>
<tr><td colspan="4">Name of Test: Advanced Voice Filtering for Rescue 21 using Machine Learning Software Efficacy Tests</td></tr>
<tr><td colspan="4">Unit Under Test (UUT):
Name: Advanced Voice Filtering for Rescue 21 using Machine Learning System
Part Number: n/a
Serial Number: n/a</td></tr>
<tr><td colspan="2">Results (Pass/Fail):</td><td colspan="2">Date of Test:</td></tr>
<tr><td>Recording of Test Measurement:

MFCC MSE (Clean and Output) =

MFCC MSE (Distorted and Clean) =</td><td>Requirements (SRD, with Tolerances):

$2 ||\hat{y} - y||^2 < ||x - y||^2$</td><td>Test Equipment Error:

n/a</td><td>Adjusted Test Limit:

n/a</td></tr>
<tr><td colspan="4">Computations, (Include Analyses Results, if any):

Calculate Mean-Squared Error of MFCC vectors
MFCC MSE (Output and Clean) = $||\hat{y} - y||^2$
MFCC MSE (Distorted and Clean) = $||x - y||^2$
Note: $|| \, . \, ||$ denotes Euclidean Norm</td></tr>
<tr><td colspan="4">Signatures:
Tester ______________________________

Customer ___________________________</td></tr>
</table>

9.9 Software Test Results

9.9.1 Software Compatibility Test Verification

Requirement 1.a : Software Compatibility				
Operating System	**Limit/Reference**	**Measured/Ref Value**	**Pass/Fail**	**Notes**
Windows XP	Executes a full prediction successfully	N/A	**Fail**	Tensorflow is not supported on Windows XP
Windows Vista	Executes a full prediction successfully	N/A	**Fail**	
Windows 7	Executes a full prediction successfully	Successful	**Pass**	
Windows 8	Executes a full prediction successfully	Successful	**Pass**	
Windows 10	Executes a full prediction successfully	Successful	**Pass**	

Table 24, Software Compatibility Results

9.9.2 Memory Usage Test Verification

Requirement 4.a : Memory Usage				
Sub-Assembly	Limit/Reference	Measured/Ref Value	Pass/Fail	Notes
File Input/Output	< 2.0 Gigabytes of RAM	210.492 MiB	Pass	Must load in many dependencies
Neural Network	< 2.0 Gigabytes of RAM	21.059 MiB	Pass	Most overhead from loading the network, not executing.
Wav Splitter/Splicer	< 2.0 Gigabytes of RAM	5.582 MiB	Pass	-
MFCC Converter/Inverter	< 2.0 Gigabytes of RAM	134.879 MiB	Pass	-
Overall System	< 2.0 Gigabytes of RAM	373.602 MiB	Pass	Well under the limit.

Table 25, Memory Usage Results

9.9.3 Total Memory Usage

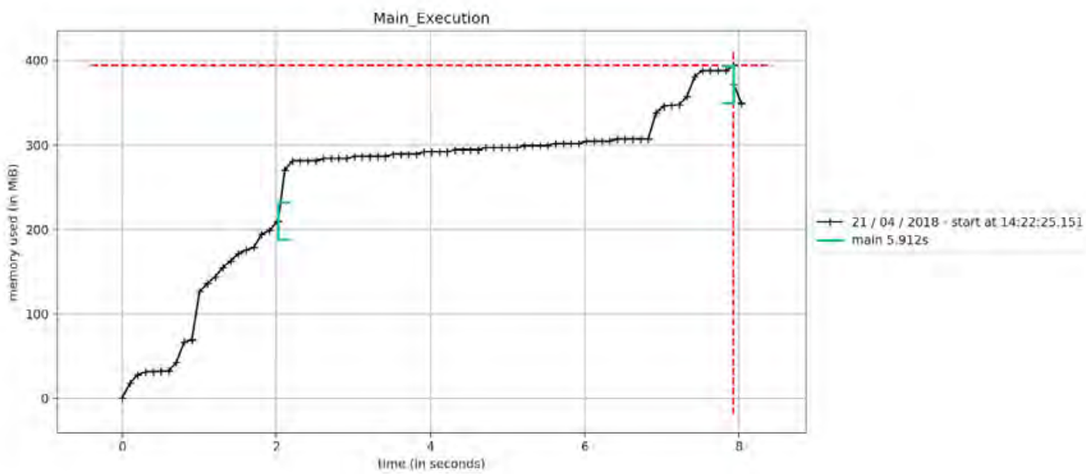


Figure 15, Total program memory usage plot.

9.9.4 Wav Splitter/Splicer and File IO Memory Usage

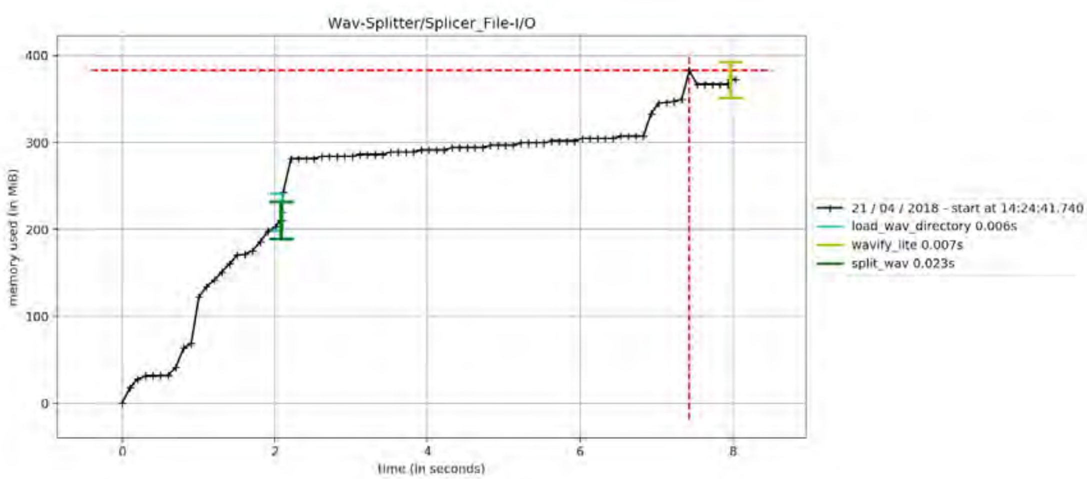


Figure 16, Wav splitter and wav splicer memory usage plot.

9.9.5 MFCC Conversion Memory Usage

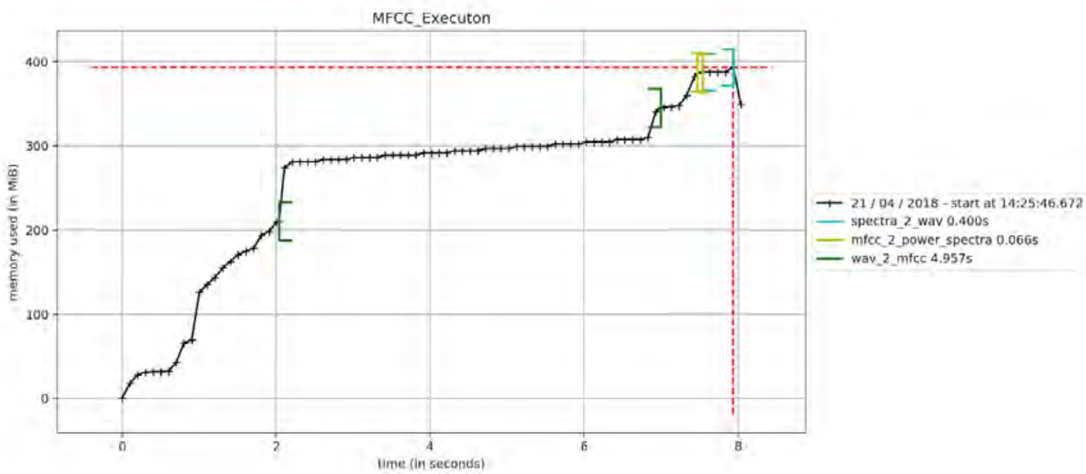


Figure 17, MFCC Conversion pipeline memory usage.

9.9.6 Neural Network Memory Usage

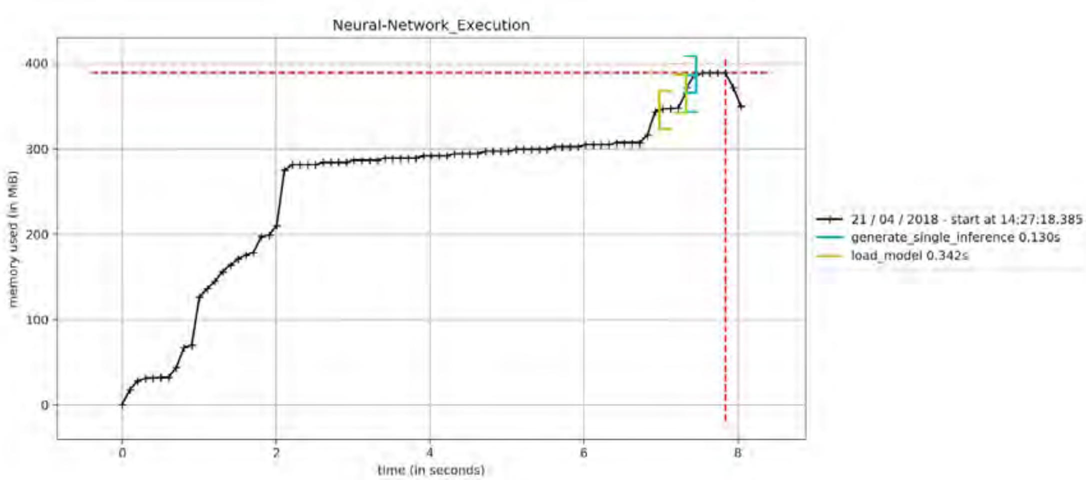


Figure 18, Neural Network memory usage plot.

9.9.7 Software Delay Verification

Requirement 4.b : Software Delay				
Sub-Assembly	**Limit/Reference**	**Measured/Ref Value**	**Pass/Fail**	**Notes**
File Input/Output	N/A	2.100 sec.	**Pass**	This sub-assembly did not have a strict requirement but we report anyways.
Neural Network	1 minutes processing per input minute	0.472 sec.	**Pass**	-
Wav Splitter/Splicer	1 minute processing per input minute	0.036 sec.	**Pass**	-
MFCC Converter/Inverter	3 minutes processing per input minute	5.423 sec.	**Pass**	Wav to MFCC conversion is significantly longer than the inversion.
Overall System	5 Minutes	8.031 sec	**Pass**	Well under the limit.

Table 26, Software Delay Results

9.9.8 Software Size Inspection Verification

Requirement 4.c : Software Size				
Sub-Assembly	**Limit/Reference**	**Measured/ Ref Value**	**Pass/ Fail**	**Notes**
File Input/Output	< 5 MegaBytes	15.0 KB	**Pass**	-
Neural Network	< 70 MegaBytes	18.4 KB	**Pass**	-
Wav Splitter/Splicer	< 15 MegaBytes	11.3 KB	**Pass**	-
MFCC Converter/Inverter	< 10 MegaBytes	15.7 KB	**Pass**	-
Executable	< 100 MegaBytes	70.3 MB	**Pass**	The executable file comprising all software
Overall System	< 100 MegaBytes	60.4 KB	**Pass**	All individual files added together

Table 27, Software Size Results

9.9.9 Software Efficacy Test Verification

Requirement 4.d : Software Efficacy				
Sub-Assembly	**Limit/Reference**	**Measured/Ref Value**	**Pass/Fail**	**Notes**
File Input/Output	N/A			
Neural Network	See Overall System			
Wav Splitter/Splicer	N/A			
MFCC Converter/Inverter	N/A			
Overall System	$2 \|\|\hat{y} - y\|\|^2 < \|\|x - y\|\|^2$ $\hat{y}$: output MFCC vector y: original target MFCC vector X: input noisy MFCC	$\|\|\hat{y} - y\|\|^2 = 39768$ $\|\|x - y\|\|^2 = 56975$	**Fail** $2 \|\|\hat{y} - y\|\|^2 > \|\|x - y\|\|^2$ 2 * 39768 > 56975 79536 > 56975	Network currently cannot generalize to large audio samples well.

Table 28, Software Efficacy Results

9.10 Signed Results Sheets

The following images are proof that our sponsor signed off on all of our requirements and verified all of the results of our test procedures.

4.8.1 Data Sheets

Advanced Voice Filtering for Rescue 21 using Machine Learning Acceptance Test Data Sheet			
Referenced ATP Paragraph Number: 4.4.1			
Analysis Referenced (for verification by T/A): N/A			
Name of Test: Software Compatibility			
Unit Under Test (UUT): **Name:** Executable Standalone Software **Part Number:** n/a **Serial Number:** n/a			
Results (Pass/Fail): Pass*		**Date of Test:** 4/24/18	
Operating System: See below	**Virtual or Hardware:**	**Where Program Failure Occured:**	**Additional Notes:**
Analyses Results, if any: Windows XP not required Windows Vista not required Windows 7 ✓ Windows 8 ✓ Windows 10 ✓			
Signatures: **Tester** ____________ **Customer** ____________			

Advanced Voice Filtering for Rescue 21 using Machine Learning Acceptance Test Data Sheet			
Referenced ATP Paragraph Number: 4.5.1			
Analysis Referenced (for verification by T/A): Memory Usage Model 100,210			
Name of Test: Memory Usage Test Procedure			
Unit Under Test (UUT): **Name:** Machine Learning Top Assembly **Part Number:** n/a **Serial Number:** n/a			
Results (Pass/Fail): Pass.		**Date of Test:** 4/24/18	
Recording of Test Measurement: Memory Usage, M = 373.6 MB	**Requirement (SRD, with Tolerances):** < 2.0 Gb ✓	**Test Equipment Error:** .2 Gb (mitigate top fluctuation by averaging)	**Adjusted Test LImit:** < 2.1 Gb
Computations, (Include Analyses Results, if any): Record process memory usage from Top over 100 forward steps (m as a function of time) as well as time elapsed. Calculate average memory usage: $M = \sum m/t$ total mem usage = 373.602 MB			
Signatures: **Tester** [signature] **Customer** [signature]			

Advanced Voice Filtering for Rescue 21 using Machine Learning Acceptance Test Data Sheet			
Referenced ATP Paragraph Number: 4.6.1			
Analysis Referenced (for verification by T/A): N/A			
Name of Test: Processing Speed Test			
Unit Under Test (UUT): **Name:**Advanced Voice Filtering for R21 using Machine Learning System **Part Number:** n/a **Serial Number:** n/a			
Results (Pass/Fail): Pass		**Date of Test:** 4/24/18	
Recording of Test Measurement: Time to finish t = 8.031 sec.	**Requirement (SRD, with Tolerances):** < 5 minutes ✓	**Test Equipment Error:** 1 second	**Adjusted Test Limit:** 1 hour
Computations, (Include Analyses Results, if any): N/A			
Signatures: Tester ___ Customer ___			

Advanced Voice Filtering for Rescue 21 using Machine Learning Acceptance Test Data Sheet			
Referenced ATP Paragraph Number: 4.7.1			
Analysis Referenced (for verification by T/A): N/A			
Name of Test: Advanced Voice Filtering for Rescue 21 using Machine Learning Software Efficacy Tests			
Unit Under Test (UUT): **Name:** Advanced Voice Filtering for Rescue 21 using Machine Learning System **Part Number:** n/a **Serial Number:** n/a			
Results (Pass/Fail): Fail		**Date of Test:** 4/24/18	
Recording of Test Measurement: MFCC MSE (Clean and Output) = 39768 MFCC MSE (Distorted and Clean) = 56975	**Requirements (SRD, with Tolerances):** $2\,\|\|\hat{y} - y\|\|^2 < \|\|x - y\|\|^2$ 79536 > 56975 X	**Test Equipment Error:** n/a	**Adjusted Test Limit:** n/a
Computations, (Include Analyses Results, if any): Calculate Mean-Squared Error of MFCC vectors MFCC MSE (Output and Clean) = $\|\|\hat{y} - y\|\|^2$ MFCC MSE (Distorted and Clean) = $\|\|x - y\|\|^2$ Note: || . || denotes Euclidean Norm			
Signatures: **Tester** ________________ **Customer** ________________			

4.a Memory Usage:
I have observed the Memory Usage Acceptance Test Procedure.

PASS / FAIL Initialis: MS

4.b Software Delay:
I have observed the Software Delay Acceptance Test Procedure.

PASS / FAIL Initialis: MS

4.c Software Size:
I have been demonstrated that the total software size is less than 100MB in total.

PASS / FAIL Initialis: MS

4.d Software Efficacy:
I have observed the Software Efficacy Acceptance Test Procedure.

PASS / FAIL Initialis: MS

Sponsor Signature: ______________ Date: 4/24/18

Final Acceptance Testing / Requirements Summary

1.a Software Compatibility:

The software has been demonstrated and tested on the following operating systems.

Windows XP	PASS / FAIL
Windows Vista	PASS / FAIL
Windows 7	PASS / FAIL
Windows 8	PASS / FAIL
Windows 10	PASS / FAIL

Initials: MS

1.b User Interface:

The software has been demonstrated to be usable via only keyboard, mouse, and monitor.

PASS / FAIL Initials: MS

3.a Testing Platform:

All training was shown to be completely conducted on AWS cloud computing instances.

PASS / FAIL Initialis: MS

3.b Data Storage:

All data was shown to be completely hosted by AWS cloud storage systems.

PASS / FAIL Initialis: MS

9.11 All Commented Code

This section shows screenshots of all major code components of the project. This includes code snapshots of :

- Wav Splitter / Splicer
- MFCC Conversion Pipeline
- Neural Network class definition
- Neural Network deployment
- Neural Network experimentation script
- Amazon Web Services storage interface
- Visualization scripts
- Usage statements

```python
def get_positions(data, slice_length, overlap):
    '''Get starting coordinates for wav slices.'''
    vals = []
    start_coord = 0

    while start_coord + slice_length < len(data):
        vals.append(start_coord)
        start_coord += (slice_length - overlap)
    return vals

def partition_wavs(wav_info, duration, overlap, wavloc=None, speed=1.0):
    '''Reads in the wav and breaks it into overlapping window wavs, and saves the
    results into a directory. These are not used immediately, but may be used by
    the autoencoder later on.

    wav_info : Can be a string name corresponding to wav data, or rate/data info.'''
    # convert seconds based on sampling rate, get initial positions, slice out
    if type(wav_info) is str:
        wav_info = wav.read(wav_info)
    rate, data = wav_info
    #rate, data = wav.read(wav_name)
    slice_length = int(round(duration * rate))
    overlap_length = int(overlap * rate)
    coordinates = get_positions(data, slice_length, overlap_length)
    wav_slices = [data[x:x+slice_length] for x in coordinates]
    #write newly created wavs to wavloc
    if wavloc is not None:
        for index, x in enumerate(wav_slices):
            wav.write(os.path.join(wavloc, str(index) + '.wav'), rate * speed, x)
    return wav_slices


def split_wav(wavloc='Wav', wav_name='traum.wav', overlap=.9, duration=1, speed = 1.0):
    '''Splits wavs based on the following parameters.

    wavloc: Output directory containing wav slices. Results will be saved in wavloc.
    wav_name: Wav file to be sliced.
    overlap: Time in seconds for overlap between each slice.
    duration: Time in seconds for duration of each sliced wav.
    speed: Playback speed. This will probably not be needed, but may be interesting for augmentation'''
    validate_settings(wavloc, wav_name, overlap, duration, speed)
    partition_wavs(wav.read(wav_name), duration, overlap, wavloc, speed)

if __name__ == '__main__':
    '''This is the same thing as split_wav using command line args for standalone testing.'''
    args = configure_args()
    partition_wavs(args.wav, args.duration, args.overlap, args.wavloc, args.speed)
```

Core functionality of Wav Splitter

```python
def examine_rate(wav_info):
    '''Checks to make sure the sampling rate of all of the wav files in the directory are the same.'''
    if not all(x[0] == wav_info[0][0] for x in wav_info):
        status('ERROR: Different sampling rates for wav files!', safe=False)
        raise SystemExit
    return wav_info[0][0]

def load_wav_directory(wavloc):
    '''Load numeric wav files from the wavloc directory, starting numbering at 0.'''
    names = [str(x)+'.wav' for x in range(len([x for x in os.listdir(wavloc) if x[-4:] == '.wav']))]
    wav_info = [wav.read(os.path.join(wavloc, x)) for x in names]
    rate = examine_rate(wav_info)
    wav_data = [x[1] for x in wav_info]
    return wav_data, rate

def wavify(filename, overlap, wavloc, recon_rate, rate):
    '''Concatenates all numerically sorted wav files into a single wav,
    subtracting out the overlapping portions.'''
    wav_data, rate = load_wav_directory(wavloc)
    sampling_ratio = recon_rate/float(rate)
    status('Reconverting with sampling ratio: ' + str(sampling_ratio) + ' [' + str(rate) + '->' + str(recon_rate) + ']')

    #compute the integer we have to slice up to (except the last wav) to ignore overlap.
    splice_index = int(rate*(len(wav_data[1])/float(rate) - overlap*rate*sampling_ratio))

    #all wavs except the last have backend overlap; combine them all, then add the last one.
    spliced = [x[:splice_index] for x in wav_data[:-1]]
    reconstructed = np.concatenate((np.concatenate((spliced)), wav_data[-1]))
    wav.write(filename, recon_rate, reconstructed)

def wavify_lite(filename, overlap, wav_array, recon_rate, rate):
    '''Concatenates all numerically sorted wav files into a single wav,
    subtracting out the overlapping portions.'''
    sampling_ratio = recon_rate/float(rate)
    status('Reconverting with sampling ratio: ' + str(sampling_ratio) + ' [' + str(rate) + '->' + str(recon_rate) + ']')

    #compute the integer we have to slice up to (except the last wav) to ignore overlap.
    splice_index = int(rate*(wav_array[0].shape[1])/float(rate) - overlap*rate*sampling_ratio)

    #all wavs except the last have backend overlap; combine them all, then add the last one & concatenate everything
    spliced = [x[:, 0:splice_index] for x in wav_array]
    spliced = [np.reshape(x, x.size) for x in spliced]
    reconstructed = np.vstack(spliced)
    reconstructed = np.reshape(reconstructed, reconstructed.size)
    wav.write(filename, recon_rate, reconstructed)

def splice_wav(overlap, filename, wavloc, recon_rate):
    '''Splices wav directory into a single wav based on the following parameters.
    overlap: Overlap time between consecutive slices. This is important for correct results.
    filename: Output spliced wav name.
    wavloc: Wav directory to be spliced
    recon_rate: splicing rate. This should be the same as the original prior to augmentation.'''
    validate_settings(wavloc, filename)
    wavify(filename, overlap, wavloc, recon_rate)
```

Core functionality of wav splicer

```python
import boto3

token = raw_input('MFA Token: ')
upload_file = raw_input('Path to file you want uploaded: ')

sts_client = boto3.client('sts')

session_token_response = sts_client.get_session_token(
                            DurationSeconds=900, #15 min
                            SerialNumber= <<<<<FILL IN WITH YOUR MFA DEVICE ARN>>>>>, #MFA ARN (can be found under MFA devies in IAM Services)
                            TokenCode=token #Token from MFA device
                        )

print(session_token_response)
print('\n\n')

ACCESS_KEY = session_token_response['Credentials']['AccessKeyId']
SECRET_ACCESS_KEY = session_token_response['Credentials']['SecretAccessKey']
SESSION_TOKEN = session_token_response['Credentials']['SessionToken']

print(ACCESS_KEY + '\n')
print(SECRET_ACCESS_KEY + '\n')
print(SESSION_TOKEN + '\n')

s3_client = boto3.client(
                'S3',
                aws_access_key_id=ACCESS_KEY,
                aws_secret_access_key=SECRET_ACCESS_KEY,
                aws_session_token=SESSION_TOKEN
                )

response = s3_client.list_buckets()

for bucket in response['Buckets']:
    print(bucket['Name'] + '\n')

s3_client.upload_file(upload_file, 'uofacapstone1718-test', upload_file)
```

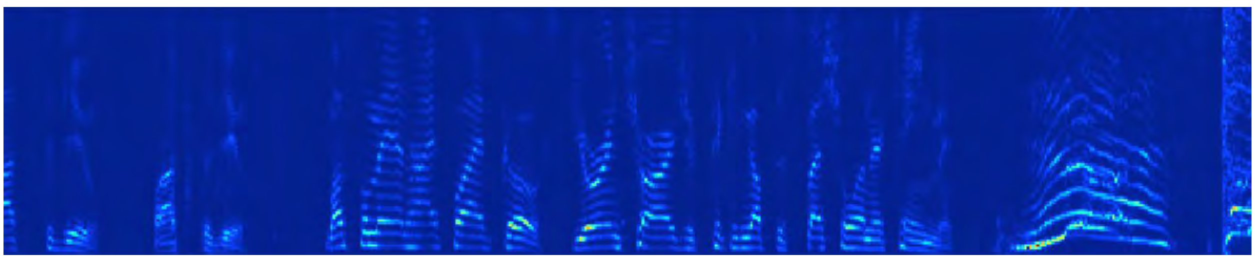
Core functionality of *S3 Data upload script*

```python
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import cm
import scipy.io.wavfile as wav
from wav_splitter import split_wav
import Data_Utils

#retrieve the training data
train_x, train_x, train_y, test_y = Data_Utils.generate_nn_sets()

displayMatrix = train_x[:, :]
fig, ax = plt.subplots()
mfcc_data = np.swapaxes(displayMatrix[:,:], 1,0) #plot left->right instead of down->up.
cax = ax.imshow(mfcc_data, interpolation='nearest', cmap=cm.jet, origin='lower', aspect='auto')
ax.set_title('MFCC Tile View')


#Showing mfcc_data
plt.show()
#Showing mfcc_feat
plt.title('Individual MFCC Lines')
plt.plot(displayMatrix)
plt.show()
```



Core functionality of matrix plotting script (MFCCs and spectrograms) and sample spectrogram plot

```python
import Data_Utils
from DeepNet import DeepNet

MODEL = 'DDAE'
#different optimizers to be tested
optimizer = ['rmsprop_momentum', 'momentum_optimizer', 'adam_optimizer']
#different learning rates to be tested
learning_rate = [0.001, 0.0001, 0.00001, 0.000001, 0.0000001]
#different alpha coefficients for momentum
momentum = [0.3, 0.5, 0.7]
#different activation functions to be teseted
activation = ['sigmoid', 'relu', 'tanh']

#file to save results to
results_file = 'results_4-12-18.txt'

w, x, y, z = Data_Utils.generate_nn_sets()

for act in activation:
    for opt in optimizer:
        for mom in momentum:
            for lr in learning_rate:
                denoising_network = DeepNet(model_architecture=MODEL, activation_function=act,
                                loss_function='mean_squared', batch_size=120, optimizer_function=opt,
                                learning_rate=lr, num_epochs=30, momentum=mom,
                                save_snapshots=False, dynamic_lr=True)
                denoising_network.fit_model(w, y, x, z)
                denoising_network.export_loss_results(results_file)

print('Experiment Done!')
```

Core functionality of neural network parameter exploration script

```python
def main():
    #Should NOT be settable; this needs to match what the NN trained on, but it's nicer to have access to it here.
    NUM_MFCCS = 125

    # generate user specified input and output files from command line
    args = configure_args()

    # Split input file
    split_wav(wavloc='tmp_audio', wav_name=args.in_file, overlap=0, duration=5, speed=1)

    # Convert split files into mfcc arrays
    mfccs, spec, phase, rate, frame = wav_2_mfcc(wavloc='tmp_audio', mfcc_num = NUM_MFCCS, fourier_points = args.fourier_point

    if args.only_convert:
        # Pass mfcc array through DeepNet
        network_input = np.array(mfccs)
        network_input = np.reshape(network_input,
                                   (network_input.shape[0] * network_input.shape[2], network_input.shape[1]))

        DDRAE = DeepNet(model_architecture='DDAE', activation_function='sigmoid',
                        loss_function='mean_squared', optimizer_function='rmsprop_momentum',
                        learning_rate=0.000001,
                        model_path='tmp/DDAE/sigmoid-rmsprop_momentum-mean_squared-lr0.001-mom0.5/model-0.ckpt')
        DDRAE.load_model(DDRAE.model_path, network_input.shape[1])
        inference = DDRAE.generate_single_inference(network_input)
    else:
        inference = mfccs

    # Reconstruct audio from mfccs
    reconstruction_spec = mfcc_2_power_spectra(inference, rate,  mfcc_num = NUM_MFCCS, fourier_points = args.fourier_points)
    #reconstruction_spec = np.transpose(inference, (0, 2, 1))
    spectra_2_wav(args.out_file, reconstruction_spec, args.disable_lsee, rate, frame, frame_overlap=0.01,
                  phase_spectral_frames=phase, fourier_points = 512)
```

```
MacBook-Pro-5:Wav-Processor- Brooks$ python RunDeepNet.py  -h
usage: RunDeepNet.py [-h] [-i] [-o] [-c] [-dl] [-f]

Denoises raw input wav files.

optional arguments:
  -h, --help            show this help message and exit
  -i , --in_file        Input noisy wav file to be denoised
  -o , --out_file       Output denoised wav file
  -c, --only_convert    Only convert and reconstruct audio from MFCCs without
                        NeuralNet
  -dl, --disable_lsee   Disables the LSEE algorithm for phase reconstruction
  -f , --fourier_points
                        Number of points used in the fourier analysis
```

Core functionality of top module wrapping deployable neural network and command line help

```python
token = raw_input('MFA Token: ')
sts_client = boto3.client('sts')
session_token_response = sts_client.get_session_token(
                            DurationSeconds=10800, #15 min
                            SerialNumber= 'arn:aws:iam::814304444943:mfa/samjgianelli@email.arizona.edu', #MFA ARN (can be found under MFA devies in IAM Services)
                            TokenCode=token #Token from MFA device
                        )
ACCESS_KEY = session_token_response['Credentials']['AccessKeyId']
SECRET_ACCESS_KEY = session_token_response['Credentials']['SecretAccessKey']
SESSION_TOKEN = session_token_response['Credentials']['SessionToken']

s3_client = boto3.client('s3',aws_access_key_id=ACCESS_KEY,aws_secret_access_key=SECRET_ACCESS_KEY,aws_session_token=SESSION_TOKEN)

BUCKET_NAME = 'uofacapstone1718-test'
RAW_DATA_DIRS = ['audio samples 11_15_17']

response = s3_client.list_objects(Bucket=BUCKET_NAME)
DATA_DIR = '/home/ec2-user/workspace/Wav-Processor-/data/'
SPLIT_DIR = '/home/ec2-user/workspace/Wav-Processor-/split/'
NUMPY_DIR = '/home/ec2-user/workspace/Wav-Processor-/numpy_arrays/'
PROCESSED_DIR = '/home/ec2-user/workspace/Wav-Processor-/numpy_arrays/'
UPDATED_DIR = PROCESSED_DIR+'updated/mfcc125/'

for obj in response['Contents']:
    if obj['Key'][-4:] == '.wav' and str(obj['Key'].split('/')[0]) in RAW_DATA_DIRS:
        #if obj['Key'][-4:] == '.npy' and str(obj['Key'][:len(RAW_DATA_DIRS[0])]) in RAW_DATA_DIRS:
        filePath = obj['Key'].split(os.sep)
        fileName = str(filePath[-1])

        if fileName[:-4]+'.npy' in os.listdir(UPDATED_DIR):
            print 'Pass'

        print '\n' + fileName
        s3_client.download_file(BUCKET_NAME, obj['Key'], DATA_DIR+'_'+fileName)

        # Create a folder for all wav partitions of input wav file if not already made
        if not os.path.isdir(SPLIT_DIR + fileName):
            os.mkdir(SPLIT_DIR + fileName)

        # split the main wav file and save paritions in split dir, delete main wave when finished
        split_wav(wavloc=SPLIT_DIR+fileName, wav_name=DATA_DIR+'_'+fileName, overlap=.0, duration=5, speed = 1.0)
        os.remove(DATA_DIR+'_'+fileName)

        # convert a split wavs to mfcc vectors, delete split directory to save space
        wavloc = SPLIT_DIR+fileName
        num_filterbanks = 125
        N = 512
        mfccs, power_spectral_frames, phase_spectral_frames, rate, frame_length = wav_2_mfcc(wavloc, mfcc_num=num_filterbanks, fourier_points=N)
        shutil.rmtree(SPLIT_DIR+fileName)

        #convert mfcc list to numpy array for saving
        mfcc_vector = np.array(mfccs).astype('float')
        np.save(UPDATED_DIR+fileName[:-4], mfcc_vector)
```

Core functionality of script for downloading and formatting data in S3 to be compatible with the neural network

```python
def generate_nn_sets():
    '''Generate test data. This needs to be updated for the data on S3.'''
    mfccs = np.load('master_mfcc_40.npy')
    mfccs = mfccs[:,:,:,:]
    np.transpose(mfccs, (0,2,1,3))
    noisy_mfccs = mfccs[:,:,:,1]
    clean_mfccs = mfccs[:,:,:,0]

    train_ind, test_ind = randomize_data_indeces(mfccs.shape[0], 0.9)
    print(noisy_mfccs.shape)
    x_train = flatten_cube(noisy_mfccs[train_ind,:,:])
    x_test = flatten_cube(noisy_mfccs[test_ind,:,:])

    y_train = flatten_cube(clean_mfccs[train_ind,:,:])
    y_test = flatten_cube(clean_mfccs[test_ind,:,:])

    return x_train, x_test, y_train, y_test

def randomize_data_indeces(size, percent_training=0.75):
    '''Generates a exclusive lists of indeces to paritition training and test data '''
    ind = np.array(range(0, size))
    #np.random.shuffle(ind)
    train_ind = np.array(ind[:int(size*percent_training)])
    test_ind  = np.array(ind[int(size*percent_training):])
    return train_ind, test_ind

def flatten_cube(x):
    '''Fit a cube (taken from MFCC converter) to a matrix that can be analyzed by
    the neural network. This should be the same for both DDAE and DRDAE.'''
    x_shape = x.shape
    return np.reshape(x, (x_shape[0] * x_shape[2], x_shape[1]))
```

Core functionality of script for formatting data to be trained/tested on

```python
# Configurable Constructor
def __init__(self, model_name='R21_NN', activation_function='sigmoid', optimizer_function='adam_optimizer',
        loss_function='squared_error', learning_rate=0.00005, batch_size=64, num_epochs=30, epoch_step=1,
        momentum=0.5, model_architecture='DDAE', num_neurons=500, data_size=40, dynamic_lr=False,
        enable_dropout=False, model_path=None, save=False, load=False, save_snapshots=False):

    self.model_name = model_name
    self.sess = None
    self.saver = None

    self.activation_function = activation_function
    self.activation = None
    self.optimizer_function = optimizer_function
    self.optimizer = None
    self.loss_function = loss_function
    self.cost = None

    self.learning_rate = learning_rate
    self.batch_size = batch_size
    self.num_epochs = num_epochs
    self.epoch_step = epoch_step
    self.momentum = momentum

    self.model_architecture = model_architecture

    self.h1_neurons = num_neurons
    self.h2_neurons = num_neurons
    self.h3_neurons = num_neurons
    self.response_size = data_size

    self.layer_1_res = None
    self.layer_2_res = None
    self.layer_3_res = None

    self.weights = None
    self.biases  = None

    self.input_placeholder         = None
    self.output_placeholder        = None
    self.learning_rate_placeholder = None
    self.keep_prob_placeholder     = None
    self.output                    = None

    self.dynamic_lr = dynamic_lr
    self.loss_results = []
    self.dropout = enable_dropout

    self.model_path = self.generate_model_path() if model_path is None else model_path
    self.save = save
    self.load = load
    self.save_snapshots = save_snapshots

    self.recurrence_length = 3 if self.model_architecture is 'DDRAE' else None
    self.validate = self.validate_step_RNN if self.model_architecture is 'DDRAE' else self.validate_step_NN
    self.train = self.train_step_RNN if self.model_architecture is 'DDRAE' else self.train_step_NN
```

Configurable constructor for building a highly customizable neural network class

```python
def DDAE_model(self):
    '''Defines the DDAE model architecture as well as the activation functions used at each layer
        Activation Functions:
                - sigmoid (default)
                - relu
                - tanh
                - none
    '''
    with tf.name_scope('hidden'):
        def activation(x): return x

        if   self.activation_function == 'sigmoid':
            activation = tf.sigmoid

        elif self.activation_function == 'relu':
            activation = tf.nn.relu
        elif self.activation_function == 'tanh':
            activation = tf.tanh

        elif self.activation_function == 'none':
            activation = activation
        # Default activation function is sigmoid
        else:
            activation = sigmoid

        self.layer_1_res = activation( tf.add(tf.matmul(self.input_placeholder, self.weights['hidden_layer_1']),
                                       self.biases['hidden_layer_1']))
        if self.dropout:
            layer_2_input = tf.nn.dropout(self.layer_1_res, self.keep_prob_placeholder)
        else:
            layer_2_input = self.layer_1_res

        self.layer_2_res = activation( tf.add(tf.matmul(layer_2_input, self.weights['hidden_layer_2']),
                                       self.biases['hidden_layer_2']))

        if self.dropout:
            layer_3_input = tf.nn.dropout(self.layer_2_res, self.keep_prob_placeholder)
        else:
            layer_3_input = self.layer_2_res

        self.layer_3_res = activation( tf.add(tf.matmul(layer_3_input, self.weights['hidden_layer_3']),
                                       self.biases['hidden_layer_3']))

        if self.dropout:
            output_input = tf.nn.dropout(self.layer_3_res, self.keep_prob_placeholder)
        else:
            output_input = self.layer_3_res

        self.output = tf.matmul(output_input, self.weights['output_layer'])

        status = 'Activation function:\t%s' %(self.activation_function)
        log.status(status, True)
        log.status('Dropout:\t\tEnabeled', True) if self.dropout else log.status('Dropout:\t\tDisabeled', True)
```

```python
def DDRAE_model(self):
    '''Defines the DDRAE model architecture as well as the activation functions used at each layer'''
    with tf.name_scope('hidden'):

        #only use feedforward on the first step
        offset = int(self.recurrence_length/2)
        data_current = self.input_placeholder[offset:offset+1, :]

        self.layer_1_res = tf.sigmoid( tf.add(tf.matmul(data_current, self.weights['hidden_layer_1']),
                                       self.biases['hidden_layer_1']))
        #recurrence_input = tf.placeholder(tf.float32, shape=[None, self.response_size, 3], name='input_layer')

        # reshape the data to be [? * respons_size, recurrence_length]. recurrence_length is usually 3.
        recurrence_reshaped = tf.reshape(self.input_placeholder, [-1, self.recurrence_length])

        # split into a list of length recurrence_length, each entry is [? x response_size].
        # Stack to make this a recurrence_length x ? x response_size tensor.
        recurrence_input = tf.stack(tf.split(recurrence_reshaped, self.recurrence_length, 1))
        recurrence_input = tf.reshape(recurrence_input, [self.recurrence_length, -1, self.response_size])

        # define the recurrence. Since the order of our dims is [time, batch, reponse_size],
        # time_major is true (first entry instead of batch).
        rnn_cell = tf.nn.rnn_cell.BasicRNNCell(num_units=500, activation=tf.sigmoid)
        rnn_outputs, states = tf.nn.dynamic_rnn(cell=rnn_cell, inputs=recurrence_input,
                                                dtype=tf.float32, time_major=True)

        # take the middle response. This is the one corresponding to the label.
        rnn_out = rnn_outputs[1]

        #weights are [RNN neurons x 40], giving a matmul dim of [? x 40].
        self.layer_3_res = tf.sigmoid(tf.add(tf.matmul(rnn_out, self.weights['hidden_layer_3']),
                                      self.biases['hidden_layer_3']))
        self.output = tf.matmul(self.layer_3_res, self.weights['output_layer'])
```

Autoencoder architectures which can be built by the configurable constructor (with and without recurrence)

```
def construct_model(self, data_size):
    '''helper function to construct all parameters of the network and save to object parameters'''
    self.response_size = data_size
    self.generate_placeholders()
    self.weights, self.biases = self.generate_model()
    self.define_architecture()
    self.create_cost_node()
    self.create_optimization_node()
    log.status('=== Model Constructed =================')
```

```
def generate_placeholders(self):
    '''Generates the input and output placeholders for the netowrk'''
    self.input_placeholder   = tf.placeholder(tf.float32, shape=[None, self.response_size], name='input_layer')
    self.output_placeholder  = tf.placeholder(tf.float32, shape=[None, self.response_size], name='output_layer')
    self.learning_rate_placeholder = tf.placeholder(tf.float32, shape=[], name='learning_rate')
    self.keep_prob_placeholder     = tf.placeholder(tf.float32, name='dropout_prob')

def generate_model(self):
    '''Generates the weights and biases vectors based on specified layer sizes'''
    weights = {'hidden_layer_1' : tf.Variable(tf.random_normal([self.response_size, self.h1_neurons]),
                                  name='hidden_layer_1_weights'),
               'hidden_layer_2' : tf.Variable(tf.random_normal([self.h1_neurons,    self.h2_neurons]),
                                  name='hidden_layer_2_weights'),
               'hidden_layer_3' : tf.Variable(tf.random_normal([self.h2_neurons,    self.h3_neurons]),
                                  name='hidden_layer_3_weights'),
               'output_layer'   : tf.Variable(tf.random_normal([self.h3_neurons,    self.response_size]),
                                  name='output_weights')}

    biases =  {'hidden_layer_1' : tf.Variable(tf.random_normal([self.h1_neurons]),    name='hidden_layer_1_bias'),
               'hidden_layer_2' : tf.Variable(tf.random_normal([self.h2_neurons]),    name='hidden_layer_2_bias'),
               'hidden_layer_3' : tf.Variable(tf.random_normal([self.h3_neurons]),    name='hidden_layer_3_bias'),
               'output_layer'   : tf.Variable(tf.random_normal([self.response_size]), name='output_bias')}

    return weights, biases

def define_architecture(self):
    '''Defines which architecture is used for the Network
    Options:
               - DDAE  (Deep Denoising Autoencoder)
               - DDRAE (Deep Denoising Recurrent Autoencoder)
    '''
    status = 'Network Architecture:\t%s' %(self.model_architecture)
    log.status(status, True)

    if  self.model_architecture == 'DDAE':
        self.DDAE_model()
    elif self.model_architecture == 'DDRAE':
        self.DDRAE_model()

    # Default architecture (DDAE)
    else:
        log.status('WARNING: USING DEFAULT ARCHITECTURE', safe = False)
        self.DDAE_model()
```

Wrapper method for building a new neural network model and select methods for demonstrating how the customizable parameters are set by the initializer

```python
def fit_model(self, training_noisy, training_clean, testing_noisy, testing_clean):
    '''Main training interface with the model.  Input noisy/clean training and testing
    data to train the model with.  Model will train for pre-specified number of epochs
    and periodically output test/validation loss'''
    self.construct_model(training_noisy.shape[1])
    self.sess = tf.Session()

    self.model_initializer(self.load)
    self.train_model(training_noisy, training_clean, testing_noisy, testing_clean)
    self.saver.save(self.sess, self.model_path)
    print('Model Saved to: '+self.model_path)

def train_model(self, training_noisy, training_clean, testing_noisy, testing_clean):
    '''method responsible for training.  Iterates through prespecified number of epochs
    running train_step with batched data'''
    log.status('=== Begin Training ==================')

    validation_loss_list = []

    validation_loss = self.validate(testing_noisy, testing_clean)
    validation_loss_list.append(validation_loss)
    print("Epoch 0:\t%f" %(validation_loss))

    for epoch in range(self.num_epochs):
        self.train(training_noisy, training_clean)

        if epoch % self.epoch_step == 0:
            validation_loss = self.validate(testing_noisy, testing_clean)
            validation_loss_list.append(validation_loss)
            if self.dynamic_lr:
                self.update_learning_rate(validation_loss_list[-1], validation_loss_list[-2])
            print("Epoch %d:\t%f" %(epoch+1, validation_loss))

            if self.save_snapshots:
                self.saver.save(self.sess, self.model_path[:-5]+str(epoch))

    self.loss_results = validation_loss_list
    log.status('=== Done Training ===================')


#training and testing for neural network (no recurrence)
def train_step_NN(self, noisy_data, clean_data):
    '''batches the input data to be ran within the tensorflow session'''
    batch_starts = range(0, len(noisy_data), self.batch_size)
    batch_ends = [i + self.batch_size for i in batch_starts]

    #train epoch
    for start, stop in zip(batch_starts, batch_ends):
        x_batch =np.array(noisy_data[start:stop, :])
        y_batch =np.array(clean_data[start:stop, :])
        tf_feed = {self.input_placeholder: x_batch, self.output_placeholder: y_batch,
                   self.learning_rate_placeholder: self.learning_rate,
                   self.keep_prob_placeholder: 0.75}

        _,c = self.sess.run([self.optimizer, self.cost], feed_dict=tf_feed)
```

```python
def validate_step_NN(self, noisy_data, clean_data):
    '''method occasionally ran during training to validate the training loss
    this dataset is completely separate from the training dataset'''
    tf_feed = {self.input_placeholder: noisy_data, self.output_placeholder: clean_data,
               self.keep_prob_placeholder: 1.0}
    result = self.sess.run([self.cost], feed_dict=tf_feed)
    return result[0]


#training and testing for neural network (with recurrence)

def train_step_RNN(self, noisy_data, clean_data):
    '''batches the input data to be ran within the tensorflow session'''
    offset = int(self.recurrence_length/2)
    batch_starts = range(0, len(noisy_data), self.batch_size * self.recurrence_length)
    batch_ends = [i + self.batch_size*self.recurrence_length for i in batch_starts]

    #train epoch
    for start, stop in zip(batch_starts, batch_ends):
        if stop > noisy_data.shape[0]:
            log.status('NOTE: I am exiting training one batch early to keep divisibility by sequence size.')
            break
        x_batch =np.array(noisy_data[start:stop, :])
        y_batch =np.array(clean_data[start+offset:start+offset+1, :])
        tf_feed = {self.input_placeholder: x_batch, self.output_placeholder: y_batch,
                   self.learning_rate_placeholder: self.learning_rate,
                   self.keep_prob_placeholder: 0.75}

        _,c = self.sess.run([self.optimizer, self.cost], feed_dict=tf_feed)

def validate_step_RNN(self, noisy_data, clean_data):
    '''method occasionally ran during training to validate the training loss
    this dataset is completely separate from the training dataset.'''
    result = 0
    offset = int(self.recurrence_length/2)

    for index in range(offset, noisy_data.shape[0] - offset):
        #slice up to 2 because we want length of 3 and python slicing is non inclusive.
        #pass in the middle label, because that's where the recurrence is.
        tf_feed = {self.input_placeholder: noisy_data[index-offset:index+offset+1, :],
                   self.output_placeholder: clean_data[index:index+1, :]}

        result += self.sess.run([self.cost], feed_dict=tf_feed)[0]
    result = result / len(range(1, noisy_data.shape[0] - 1))
    return result
```

Train/test functions for both neural network architectures

```python
def generate_single_inference(self, data):
    '''Single inference method to input one feature vector and return a single prediction.
    Mostly used for debugging as it has no practicle purpose in deployment'''
    prediction = self.output.eval(session=self.sess, feed_dict={self.input_placeholder : data})
    print('=Network===')
    prediction = np.reshape(prediction, (-1,data.shape[1],332))
    return prediction

def generate_batch_inference(self, data):
    '''Main deployement method after training has finished.
    Accepts a a matrix of data to generate inferences upon.
    Return a matrix of inferences that can be reassmebled into a wav file.'''

    results = np.empty((0,self.response_size), float)

    for i in range(1,data.shape[0]-1):
        tf_feed = {self.input_placeholder: data[i-1:i+2,:]}
        prediction = self.output.eval(session=self.sess, feed_dict=tf_feed)
        results = np.vstack((results, prediction))
    return results
```

Prediction methods for applying a trained neural network model to new MFCCs/spectrograms

```python
def wav_2_mfcc(wavloc, frame_duration=0.025, frame_overlap=0.01, fourier_points=512, mfcc_num=40):

    # spectral parameters
    num_filterbanks = mfcc_num
    cep_lifter = 2 * mfcc_num

    # LSEE parameters - NOTE: LSEE is currently disabled as it may not be needed!
    max_iterations = 100
    LSEE_stop_error = 100  # in dB
    LSEE_row_chunk_size = 20  # note: keep below 100

    # 1) load list of wav files (~5 seconds each)
    status('Loaded wav directory contents...')
    wavs, rate = load_wavs(wavloc)
    status('Loaded ' + str(len(wavs)) + ' wav files with sampling rate: ' + str(rate))

    # do float operations. Don't forget to return to int16 or else get wrecked
    wavs_float = [1.0 * wav for wav in wavs]

    # if wavs are in stereo (or more channels), turn to mono
    if (wavs_float[0].ndim > 1):
        wavs_float = [wav.sum(axis=1) / wavs_float[0].ndim for wav in wavs_float]

    # 3) frame signal
    status('Framing signals...')
    framed_wavs, frame_length, rate = framify_wavs(wavs_float, rate)

    # 3.5) interpolate linearly to make our vectors of length 512 for the fourier transform (doesn't affect audio quality
    # nor does it affect the spectrogram!
    status('Interpolating linearly...')
    interp_framed_wavs = [scipy.signal.resample(x, fourier_points, axis=1) for x in framed_wavs]

    # 4) Windowing function
    # windowed_wavs: each row vector is a chunk of wav
    status('Applying Hamming Window...')
    windowed_wavs = [x * np.hamming(len(x[0][:])) for x in interp_framed_wavs]

    # 5) Fourier Transform (real valued array) -> power spectra
    # use rfft because we assume mel filtering is symmetric about the DC frequency (f=0)
    status('Evaluating power spectra...')
    power_spectral_frames = [np.fft.rfft(frame_wav, fourier_points)/float(fourier_points) for frame_wav in windowed_wavs]
    phase_spectral_frames = [np.angle(ps_frame)      for ps_frame in power_spectral_frames]
    power_spectral_frames = [np.absolute(ps_frame)   for ps_frame in power_spectral_frames]

    # 6) Filter Bank
    status('Computing spectrograms (filter banks)...')
    filter_banks = generate_filters(rate, fourier_points, num_filterbanks)
    spectrograms = [apply_filter(ps_frame, filter_banks) for ps_frame in power_spectral_frames]
    # Note: now, column vectors are spectrograms, NOT row vectors

    # 7) MFCCs
    status('Computing MFCCs...')
    mfccs = [MFCC(x, num_filterbanks, cep_lifter) for x in spectrograms]

    #return MFCCs
    return mfccs, power_spectral_frames, phase_spectral_frames, rate, frame_length
```

Core wrapper functions for converting wav files to MFCCs

```python
def mfcc_2_power_spectra(mfccs, rate, fourier_points=512, mfcc_num=40):
    N = fourier_points
    num_filterbanks = mfcc_num
    cep_lifter = 2 * mfcc_num

    # -7) Inverting MFCCs
    status('Inverting MFCCs...')
    spectrograms = [inv_MFCC(mfcc, num_filterbanks, cep_lifter) for mfcc in mfccs]

    # -6) Inverting filter bank
    status('Inverting spectrograms (filter banks)...')
    filter_banks = generate_filters(rate, N, num_filterbanks)
    inv_filter_banks = np.linalg.pinv(filter_banks)
    power_spectral_frames = [invert_apply_filter(spec, inv_filter_banks) for spec in spectrograms]
    return power_spectral_frames

def spectra_2_wav(outfile, power_spectral_frames, run_LSEE, rate, frame_length, phase_spectral_frames=None, frame_overlap=0.01, fourier_points=512):
    N = fourier_points
    max_iterations = 100
    LSEE_stop_error = 100 # dB
    LSEE_row_chunk_size = 20

    # -5) LSEE: an algorithm to get us from spectra to windowed_wavs
    status('Inverting power spectra...')
    if phase_spectral_frames is None:
        print('    No phase supplied, defaulted to running LSEE')
        windowed_wavs2 = [LSEE(ps_frame, N, max_iterations, LSEE_stop_error, LSEE_row_chunk_size) for ps_frame in power_spectral_frames]

    else:
        if run_LSEE == 1:
            status('    Running LSEE with phase...')
            psf = [(np.exp(1j * phase_spectral_frames[i]) * power_spectral_frames[i]) for i in
                   range(len(power_spectral_frames))]
            windowed_wavs2 = [LSEE(ps_frame, N, max_iterations, LSEE_stop_error, LSEE_row_chunk_size) for ps_frame in psf]
        else:
            status('    No LSEE, just phase...')
            windowed_wavs2 = [np.real(np.fft.irfft(np.exp(1j * phase_spectral_frames[i]) * power_spectral_frames[i] * float(N)))
                              for i in range(len(power_spectral_frames))]

    # -4) invert windowing
    status('Inverting Hamming Window...')
    # frame length
    dewindowed_wavs = [x / np.hamming(len(x[0][:])) for x in windowed_wavs2]

    # -3.5) resample again to return to row vectors of length 400
    status('Backwards interpolating linearly...')
    interp_dewindowed_wavs = [scipy.signal.resample(x, int(frame_length), axis=1) for x in dewindowed_wavs]
    # Convert back to int16; this is very important!
    interp_dewindowed_wavs = [np.int16(wav) for wav in interp_dewindowed_wavs]

    # -3) deframe signal
    status('De-Framing signals...')                            # 1-1 rate conversion
    wav_splicer.wavify_lite(outfile, frame_overlap, interp_dewindowed_wavs, rate, rate)
```

Core wrapper functions for converting MFCCs to wav files