CMPE492 – Senior Project II, Fall 2021

Test Plan Report

Voiceolation

Ahmad Ali Abdel Latif Amireh

Emir Yılmaz

Kemalcan Güner

Yunus Emre Günen

Supervisor: Gökçe Nur Yılmaz

Jury Members: Aslı Gençtav, Venera Adanova

Neural Network Testing

Theoretically, training an artificial neural network is accomplished by finding the optimal values for the weights and biases with tests. In supervised systems, training is a process that can be described as train-test-valid procedures. Neural network and testing is already a unified concept since a neural network model trains itself by comparing the result with labeled data. Beside, there are external evaluation methods for testing the system output such as SDR (Source-to-Distortion Ratio), SIR (Source-to-Interference Ratio), and SAR (Source-to-Artifact Ratio) in regards to our case which means audio source separation. SDR is the common measure method, it calculates the ratio of original source audio and errors such as noise, artifacts etc. The results are in decibels (dB) and the higher is confirmed as better. We will use these values for benchmarking, to compare our results and other related works on MUSDB18[1] and at SiSEC 2018[2].

Also, there is an optional testing that we can call ‘overfit testing’ to prevent overfitting while training neural networks with the dataset. We can use callback functions which are used for interfering or checking any stage of training procedure. For example in Keras, the EarlyStopping function can be used in order to avoid overfitting. This function monitors the validation and training values. When validation loss starts to rise after the number of epochs, it is a sign of the beginning of the overfitting and it means we will need to eventually stop training to avoid overfitting. Therefore, our test case would be checking if validation loss consistently increases or not.

Bottom-Up Integration Testing

For every object and function, we will test them bottom to the top. For example, we wrote a preprocess function, it calls other functions such as short-time fourier transform (STFT) and Butterworth low-pass filter functions; thus, first test the callee functions. Then, the preprocess function, after that we will test their integration and it continues on like that at every step. One of the most important advantages of bottom-up integration testing is that we can test different subsystems simultaneously. In bottom-up integration testing, when we find an error we can localize it; since we know which subprograms and modules are integrated or working up to our testing state.

[1] MUSDB18 Benchmark: https://paperswithcode.com/sota/music-source-separation-on-musdb18
[2] SiSEC MUS 2018 Evaluation Results: https://sisec18.unmix.app#/results/vocals/SDR

Unit Testing

We will test every unit such as objects and subprograms independently as possible. For example, we will use the write_soundfile function at the very end of the process. However, we will test it independent from the vocal separation by just sending time series and check whether it can convert them to sound file or not. We prefer unit testing because after testing units, there is no need to dive deep in the code again when facing an error.

Performance Testing

Volume testing will be applied to the system. Since our dataset is sizable, we need to check whether our data structures can handle overflowings or other extreme situations. For instance, if a user tries to upload high quality music with .FLAC extension, we can encounter problems converting it to .wav or even applying short-time fourier transform due to their sizes being bigger than samples that used on the training model.

Moreover, we are planning to build a website with a basic UI to make available for public use, it's performance matters. Therefore, we will test the performance, speed of our source separation model. Our goal is to separate the vocals of an average-length song in seconds.

Documentation Testing

Since we are using a lot of libraries and dependencies, it is necessary to conduct a requirement file and technical manuals to keep our software maintainable and consistent. After acquiring a stable version of the project, we will create requirements.txt and/or a pipfile to maintain libraries and configure the project. Also, we will update these files whenever there is a change.

Test Cases for the Web Application

Our possible test cases for the website except neural network and other processes' tests:

Test Case #	Description	Expected Result
1	Check the upload button when "terms of service" not accepted	Unclickable "browse" button to upload
2	Check the upload button when "terms of service" accepted	Clickable "browse" button to upload
3	Click "browse" button	Open a file explorer windows to show and upload sound files
4	Uploading a sound file	Response with two playable sound files
5	Play the sound file	Response audio output of the sound file
6	Download the sound file	Successfully open a download window and starting to download
7	Search for the last uploaded file on the server of the website	Not be able to reach that file, it must be deleted from the server after the whole process