Audio Splitter App

Split an input audio file into different channels of instruments to be used for backing tracks. Solves having to use bad quality YouTube videos as only source for backing.

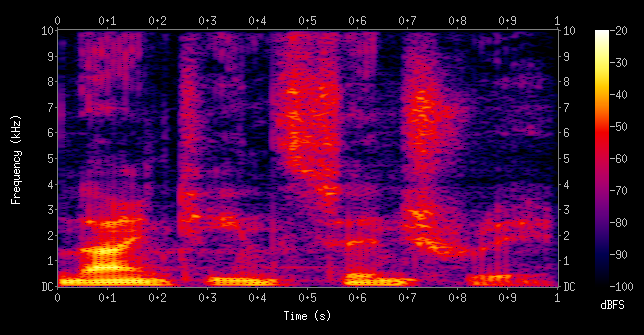
Diagram

Description automatically generated

Plan to make it work by changing input audio to a set bitrate and passing through a specific instrument neural network as a spectrogram. Run through entire spectrogram column by column and using current, before and next spectrogram values as neural network input, providing which current spectrogram values are the instrument as the output.

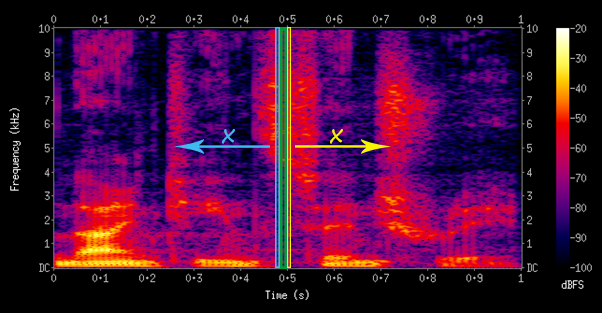
Could be acquired by audio streaming service or I could run ads and initial payment for revenue.

Spectrograms



A 3-dimensional graph used to represent audio. The x-axis is used to represent how far through the audio each column of the graph is. The y-axis is used to represent what frequency each row is. The brightness of each section represents the volume or amplitude of each section in dBFS (decibels full scale).

Neural Network Inputs



Use volumes of green row alongside volumes of previous x rows (blue) alongside volumes of next x rows (yellow).

Training Neural Network

Use gradient descent, calculating errors from sample data found here: <https://sigsep.github.io/>. Includes 9 sources to a total of 1733 samples which include: total track, bass, drums, other, and vocals.

Sub Tasks

* C++ implementation of audio files (preferably MP3) to spectrogram data, which can be saved to a spectrogram.
* Processing a full song’s spectrogram data to a specific track’s spectrogram data.
* Processing output spectrogram data to audio.
* Method of downloading audio into program by user.
* User interface.

WAV File Format Breakdown

<http://soundfile.sapp.org/doc/WaveFormat/>

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Size in Bytes** | **Example (hex)** |
| Chunk Identifier | “RIFF” | 4 | 52 49 46 46 “RIFF” |
| Chunk Size | Size of chunk after this | 4 |  |
| Format | “WAVE” | 4 | 57 41 56 45 “WAVE” |
| Sub Chunk 1 Identifier | “fmt “ | 4 | 66 6d 74 20 “fmt “ |
| Sub Chunk 1 Size | 16 | 4 | 10 00 00 00 |
| Audio Format | PCM = 1, any other values show compression | 2 | 01 00 |
| Channel Count | 1 – Mono 2 – Stereo | 2 | 02 00 |
| Sample Rate | Samples per second | 4 | 22 56 00 00 |
| Byte Rate | Sample rate \* channel count \* (bits per sample / 8) | 4 | 88 58 01 00 |
| Block Align | Bytes per sample for all channels | 2 | 04 00 |
| Bits Per Sample | Bits in a sample | 2 | 10 00 |
| Sub Chunk 2 Identifier | “data” | 4 | 64 61 74 61 “data” |
| Sub Chunk 2 Size | Samples \* channels \* (bits per sample / 8) | 4 | 00 08 00 00 |
| Data | The sound data | “Sub Chunk 2 Size” | Below |

Audio data samples – bits in order of their channels (left then right if stereo) representing amplitude at a certain time. The time **t** at sample **s** can be written as .

The wavelengths between two amplitude peak times, **t0** and **t1** can be calculated using , thus making the frequency at time equal to or .

MP3 File Format

<https://github.com/lieff/minimp3>

Using “minimp3.h” (base level) and “minimp3\_ex.h” (high level API) to read in MP3 files.

Text

Description automatically generated