UNIVERSITY OF ENGINEERING AND TECHNOLOGY TAXILA (UET)

COMPUTER SCIENCE ENGINEERING DEPARTMENT

Project

Music Source Separation using Open-Unmix PyTorch

For

Sixth Semester

INSTRUCTOR: Sir. Shahid Bhutta

Submitted by: Muhammad Subhan, Jawad Asghar, Zabiullah

Reg. number: 20-CP-77, 20-CP-04, 20-CP-95

Date: 27/06/2022

Table of Contents

MUSIC SOURCE SEPARATION USING OPEN-UNMIX PYTORCH

Problem Statement:

Music source separation is the venture of isolating individual sound sources, including vocals, drums, bass, and different gadgets, from a combined audio sign. This method is hard because of the complicated nature of musical signals and the overlapping frequency additives. However, it's far an essential mission for various packages, inclusive of audio engineering, remixing, and tune analysis.

Aims/ Objectives:

The aim of this challenge is to provide a smooth-to-use answer for track supply separation using the Open-Unmix PyTorch library. The venture ambitions to permit researchers, audio engineers, and artists to separate pop tunes into 4 stems: vocals, drums, bass, and other devices. By leveraging pre-skilled models educated on the MUSDB18 dataset, the undertaking objectives are to supply correct and reliable supply separation effects.

Abstract:

The Music Source Separation mission makes use of the Open-Unmix PyTorch library to put into effect a deep neural network model for setting apart music assets. The task gives pre-skilled models that permit users to split pop music into 4 stems, namely vocals, drums, bass, and other units. The models are skilled at the MUSDB18 dataset, which ensures their effectiveness in real-world eventualities. The mission offers a person-friendly net-based totally interface where users can add their audio files and attain separated stems as output. The separation procedure includes changing the enter audio to WAV format, applying supply separation using the Open-Unmix model, normalizing the ensuing waveforms, and saving the separated stems as person audio documents. The undertaking's code is constructed the usage of the Flask framework and PyTorch library, enabling green execution and interaction with the models.

Introduction:

Music source separation is a complex challenge that includes setting apart character sound sources, together with vocals, drums, bass, and other gadgets, from a mixed audio sign. It performs an important role in numerous domains, consisting of track production, audio engineering, and tune evaluation. The capacity to isolate and manipulate character additives of a tune recording offers outstanding creative possibilities and allows for better audio first-rate and information.

The Music Source Separation project goal is to cope with the challenges of music source separation by utilizing the Open-Unmix PyTorch library. Open-Unmix offers a deep neural network implementation particularly designed for song source separation duties. By leveraging the electricity of deep getting to know, the venture permits researchers, audio engineers, and artists to separate pop music into four wonderful stems: vocals, drums, bass, and other gadgets.

The challenge's primary objective is to offer a user-pleasant and handy solution for music source separation. It gives pre-skilled models educated on the MUSDB18 dataset, that's widely identified as a benchmark dataset for track supply separation. These fashions have been trained on a diverse variety of tune recordings and can deliver accurate and dependable separation consequences.

With the assignment's web-based totally interface, customers can effortlessly upload their audio documents and gain separated stems as output. The interface offers an intuitive and interactive experience, permitting users to visualize and listen to the original audio as well as the separated stems. This empowers customers to discover and analyze the individual components of track recordings, beginning up avenues for creative remixing, audio enhancement, and in-depth track evaluation.

The underlying methodology of the challenge involves changing the uploaded audio documents to a WAV layout, applying the Open-Unmix model to split the unique tune assets, normalizing the resulting waveforms for consistency, and saving the separated stems as a man or woman audio file. The venture leverages the Flask framework for net development and PyTorch for the green execution of the deep neural network version.

In the end, the Music Source Separation undertaking the use of Open-Unmix PyTorch affords a valuable tool for researchers, audio engineers, and artists to perform song source separation without difficulty. By harnessing the energy of deep studying and pre-educated models, the assignment permits the extraction of vocals, drums, bass, and different contraptions from combined audio recordings. This empowers customers to release the capacity of song recordings, facilitating innovative endeavours, audio enhancement, and insightful music evaluation.

Methodology:

The methodology hired inside the Music Source Separation task the usage of Open-Unmix PyTorch includes numerous key steps to gain accurate and reliable separation of song sources. The assignment leverages the talents of deep neural networks and pre-skilled fashions to address the intricate task of supply separation. The following Points offer an overview of the methodology concerned with the mission.

- ✓ **User Interface:** The project utilizes an internet-based interface built with Flask, HTML, and JavaScript. The interface allows users to add their audio documents and examine the separated stems.
- ✓ **Pre-processing:** The uploaded audio document is converted to the WAV format if it's miles in MP4 layout. Torchaudio library is used to load the waveform and pattern price of the audio.
- ✓ **Source Separation:** The Open-Unmix model is employed to split the exceptional track resources from the waveform. The model takes the waveform as input, tactics the usage of deep neural networks, and produces estimates for each supply.
- ✓ **Normalization:** The separated waveforms are normalized to ensure consistency and decorate the listening reveal in. The minimal and most values of the waveforms are determined, and the waveforms are scaled thus.
- ✓ **Saving Separated Stems:** The separated stems are stored as man or woman audio documents inside the 'stems' folder. Each stem is associated with a name (vocals, drums, bass, or other instruments) and a corresponding audio file path.
- ✓ **Output Generation:** The challenge generates a response item containing the trails to the authentic audio document and the separated stems. This response is dispatched back to the consumer interface for the show.

CODE:

The code implements the Music Source Separation task using Flask and PyTorch. It consists of capabilities for converting audio to WAV layout, separating audio assets the usage of Open-Unmix, normalizing waveforms, and dealing with file uploads. The code also defines routes for the web interface and serves the separated audio files. It leverages the touch audio and subprocess libraries for audio processing and makes use of the Open-Unmix library for supply separation.

```
from flask import Flask, request, jsonify, send_from_directory
from werkzeug.utils import secure_filename
import os
import torch
import torchaudio
import subprocess
import time

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = 'uploads'
use_cuda = torch.cuda.is_available()
device = torch.device("cuda" if use_cuda else "cpu")

def convert_to_wav(mp4_path):
    if mp4_path.endswith(".mp4"):
        wav_path = mp4_path[:-4] + ".wav"
        subprocess.run(["ffmpeg", "-i", mp4_path, wav_path])
        return wav_path
    return mp4_path

def separate_audio(waveform, sample_rate):
    from openunmix import predict
    estimates = predict.separate(
        waveform.unsqueeze(0).to(device),
        rate=sample_rate,
        device=device
    )
    return estimates

def normalize_waveform(waveform):
    min_value = torch.min(waveform)
    max_value = torch.max(waveform)
    normalized_waveform = (waveform - min_value) / (max_value - min_value)
    return normalized_waveform

@app.route('/')
def index():
    return '''
    <!DOCTYPE html>
    <html>
    <head>
        <title>Audio Separation</title>
        <style>
            body {
                font-family: Arial, sans-serif;
                margin: 0;
                padding: 0;
            }
            .container {
                max-width: 600px;
```

```
    margin: 0 auto;
    padding: 20px;
}
.title {
    font-size: 24px;
    font-weight: bold;
    text-align: center;
    margin-bottom: 20px;
}
.upload-form {
    display: flex;
    justify-content: center;
    align-items: center;
    margin-bottom: 20px;
}
.upload-input {
    border: none;
    border-radius: 4px;
    padding: 10px;
    font-size: 16px;
    margin-right: 10px;
    flex-grow: 1;
}
.upload-button {
    background-color: #4CAF50;
    color: white;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    padding: 10px 20px;
    font-size: 16px;
}
.audio-container {
    margin-bottom: 20px;
}
.audio-title {
    font-size: 18px;
    font-weight: bold;
    margin-bottom: 5px;
}
.audio-element {
    width: 100%;
    outline: none;
}
.stems-container {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
    grid-gap: 20px;
}
.stem-card {
    border: 1px solid #ddd;
    border-radius: 4px;
    padding: 10px;
    text-align: center;
    box-shadow: 0px 2px 4px rgba(0, 0, 0, 0.1);
    transition: box-shadow 0.3s ease-in-out;
}

.stem-card:hover {
    box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.2);
}
```

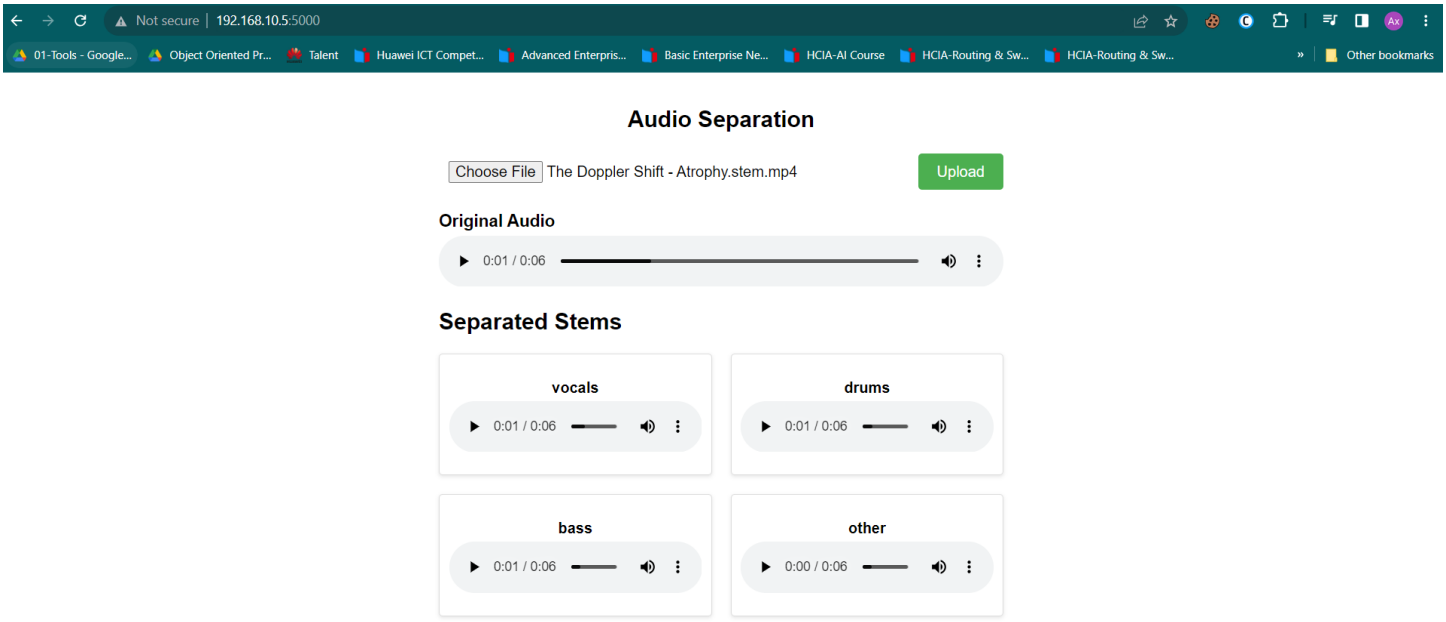
```
        .stem-name {
            font-size: 16px;
            font-weight: bold;
            margin-bottom: 5px;
        }
        .stem-audio {
            width: 100%;
            outline: none;
            margin-bottom: 10px;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1 class="title">Audio Separation</h1>
        <form class="upload-form" action="/upload" method="post" enctype="multipart/form-data">
            <input class="upload-input" type="file" name="audioFile" accept=".mp3, .wav, .mp4">
            <input class="upload-button" type="button" value="Upload" onclick="uploadFile()">
        </form>
        <div class="audio-container">
            <h2 class="audio-title">Original Audio</h2>
            <audio class="audio-element" id="originalAudio" controls></audio>
        </div>
        <h2>Separated Stems</h2>
        <div class="stems-container" id="stems"></div>
    </div>
    <script>
        function uploadFile() {
            var fileInput = document.querySelector('input[type="file"]');
            var file = fileInput.files[0];
            var formData = new FormData();
            formData.append("audioFile", file);
            var xhr = new XMLHttpRequest();
            xhr.open("POST", "/upload", true);
            xhr.onreadystatechange = function () {
                if (xhr.readyState === 4 && xhr.status === 200) {
                    var response = JSON.parse(xhr.responseText);
                    displayAudio(response);
                    displaySeparatedStems(response);
                }};
            xhr.send(formData);
        }
        function displayAudio(response) {
            var audioElement = document.getElementById("originalAudio");
            audioElement.src = response.originalAudio;
        }
        function displaySeparatedStems(response) {
            var stems = response.stems;
            var stemsContainer = document.getElementById("stems");
            stemsContainer.innerHTML = ""; // Clear the existing content
            for (var i = 0; i < stems.length; i++) {
                var stem = stems[i];
                var stemCard = document.createElement("div");
                stemCard.classList.add("stem-card");
                var stemName = document.createElement("p");
                stemName.classList.add("stem-name");
                stemName.textContent = stem.name;
                stemCard.appendChild(stemName);
                var audioElement = document.createElement("audio");
                audioElement.classList.add("stem-audio");
                audioElement.controls = true;
```

```
            audioElement.src = stem.audio;
            stemCard.appendChild(audioElement);
            stemsContainer.appendChild(stemCard);
          }}
    </script>
  </body>
  </html>
  '''

@app.route('/upload', methods=['POST'])
def upload():
  file = request.files['audioFile']
  filename = secure_filename(file.filename)
  filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
  file.save(filepath)
  wav_path = convert_to_wav(filepath)
  waveform, sample_rate = torchaudio.load(wav_path)
  if waveform.shape[0] > 1:
    waveform = torch.mean(waveform, dim=0)
  normalized_waveform = normalize_waveform(waveform)
  estimates = separate_audio(normalized_waveform, sample_rate)
  stems_folder = os.path.join(app.config['UPLOAD_FOLDER'], 'stems')
  os.makedirs(stems_folder, exist_ok=True)  # Create the 'stems' folder if it doesn't exist
  stems = []
  for target, estimate in estimates.items():
    audio = estimate.squeeze().detach().cpu().numpy()
    stem_filename = f"{filename}_{target}.wav"
    stem_filepath = os.path.join(stems_folder, stem_filename)
    torchaudio.save(stem_filepath, torch.from_numpy(audio), sample_rate)
    stem = {
      'name': target,
      'audio': f'uploads/stems/{stem_filename}'
    }
    stems.append(stem)
  response = {
    'originalAudio': f'uploads/{filename}',
    'stems': stems
  }
  # Wait for a short period to allow other processes to release the file
  time.sleep(1)
  # Remove the temporary WAV file
  os.remove(wav_path)
  return jsonify(response)
@app.route('/uploads/<filename>')
def serve_file(filename):
  return send_from_directory(app.config['UPLOAD_FOLDER'], filename)
@app.route('/uploads/stems/<filename>')
def serve_stem(filename):
  return send_from_directory(os.path.join(app.config['UPLOAD_FOLDER'], 'stems'), filename)
if __name__ == '__main__':
  app.run(host='0.0.0.0', port=5000)
```

GUI View:

The Project consists of HTML and CSS patterns for the user interface. The interface includes a report upload form, an audio player to play the authentic audio, and a section to display the separated stems. The stems are supplied in person cards, showing the name of the stem and an audio player for playback.



1. GUI interface of Music source separation Project

Conclusion:

In conclusion, the Music Source Separation undertaking utilizing Open-Unmix PyTorch provides an intuitive and client-pleasant solution for isolating special track resources from mixed audio recordings. Leveraging the electricity of deep neural networks and pre-trained models, this task empowers customers to extract stems for vocals, drums, bass, and other gadgets. Through its net-based interface, customers can effortlessly add their audio documents and visualize the separated stems, enhancing their tune production, audio engineering, and track evaluation workflows.

The effectiveness of Open-Unmix for song source separation tested in this undertaking has extensive implications for song production. By keeping apart man or woman stems, manufacturers and remixers advantage of unprecedented control and flexibility over their mixes. They can first-rate-tune the ranges, follow custom outcomes, and create specific remixes or new arrangements. This capability permits greater innovative experimentation and empowers artists to achieve their preferred sonic vision.

Furthermore, audio engineers can take advantage of this project by means of leveraging the separated stems for numerous duties. Whether it's audio healing, sound design, or submit manufacturing, getting access to isolated tune assets greatly complements the engineer's capacity to control the audio. Unwanted elements may be eliminated, precise elements may be more desirable, and the overall quality and readability of the final output can be advanced, ensuing in a greater polished and expert sound.

The Music Source Separation challenge also opens avenues for song evaluation and research. By imparting man or woman stems for distinctive devices and vocals, researchers and musicologists can delve into the intricacies of a recording. They can analyze the traits of each stem one after the other, examine the interplay between numerous elements, and gain insights into the composition and arrangement of the unique piece. This deeper expertise of the track can cause new discoveries and contribute to the advancement of tune ideas and evaluation.

In precis, the Music Source Separation venture making use of Open-Unmix PyTorch gives a user-friendly and effective solution for setting apart music assets from mixed audio recordings. Its application in track manufacturing, audio engineering, and song analysis affords better innovative possibilities, advanced audio best, and precious insights into the composition and overall performance of the tune.