EAI 6010:

APPLICATIONS OF ARTIFICIAL INTELLIGENCE

Module 4: Audio Classification with PyTorch: A Documentation

Submitted To:

Prof RJ Munthali, Faculty Lecturer

Submitted By:

Abhilash Dikshit

Academic Term: Fall 2023

Northeastern University, Vancouver, BC,Canada

Master of Professional Studies in Analytics

December 06, 2023

Title: Audio Classification with PyTorch: A Documentation

I. Introduction

This documentation outlines the work conducted in implementing an audio classification model using PyTorch based on the Microsoft tutorial. The primary objective was to understand audio data, transform sound signals into visual representations (spectrograms), and build a speech classification model capable of recognizing sounds or spoken words, specifically 'yes' and 'no' commands.

II. Dataset Selection

The dataset chosen for this project is the Speech Commands dataset provided by PyTorch. This dataset includes various spoken words and commands, but for our binary classification model, we focused on the 'yes' and 'no' classes. The decision to select these classes was based on the simplicity of the task and the availability of a well-prepared dataset for experimentation.

III. Data Preparation

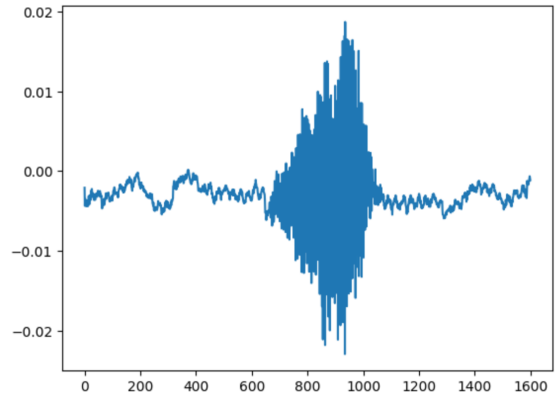
The Speech Commands dataset was directly downloaded using PyTorch's TorchAudio library. The 'yes' and 'no' classes were utilized for creating a binary classification model. Each audio sample was represented as a tensor, and the associated sample rate was 16,000 Hz. The waveform, sample rate, label, and unique identifier for each audio file were considered during the data preparation phase.

IV. Data Transformation and Visualization

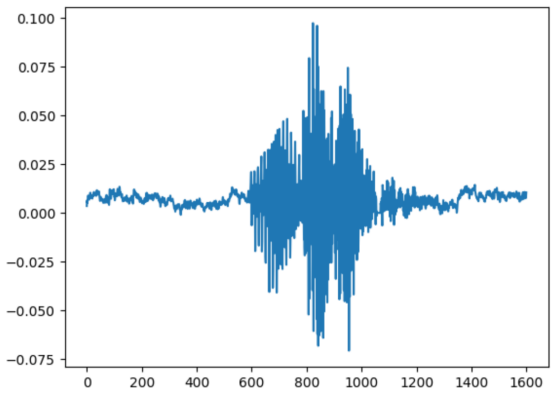
Waveform

The waveform, representing the signal visually over time, was examined for both 'yes' and 'no' commands. The waveform was visualized, and sample audio clips were played to gain insights into the structure of the audio signals.

```
Waveform: tensor([[-0.0028, -0.0054, -0.0034,  ..., -0.0011, -0.0013, -0.0014]])
Sample rate: 16000
Labels: yes

Shape of transformed waveform: torch.Size([1, 1600])
Sample rate: 1600.0
```



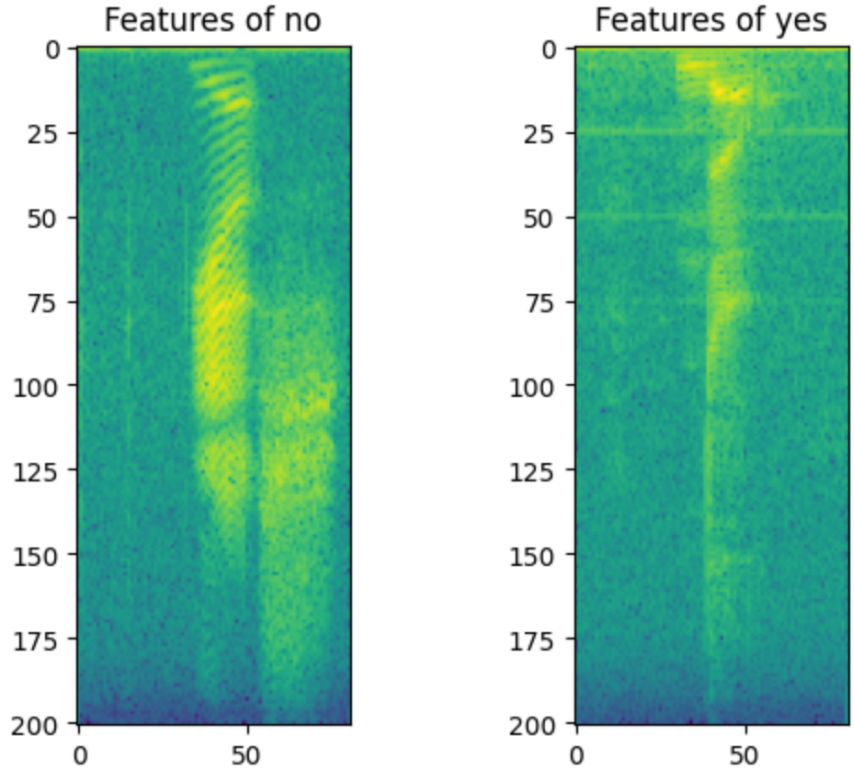

```
Waveform: tensor([[0.0072, 0.0061, 0.0055,  ..., 0.0098, 0.0085, 0.0092]])
Sample rate: 16000
Labels: no

Shape of transformed waveform: torch.Size([1, 1600])
Sample rate: 1600.0
```



Spectrogram

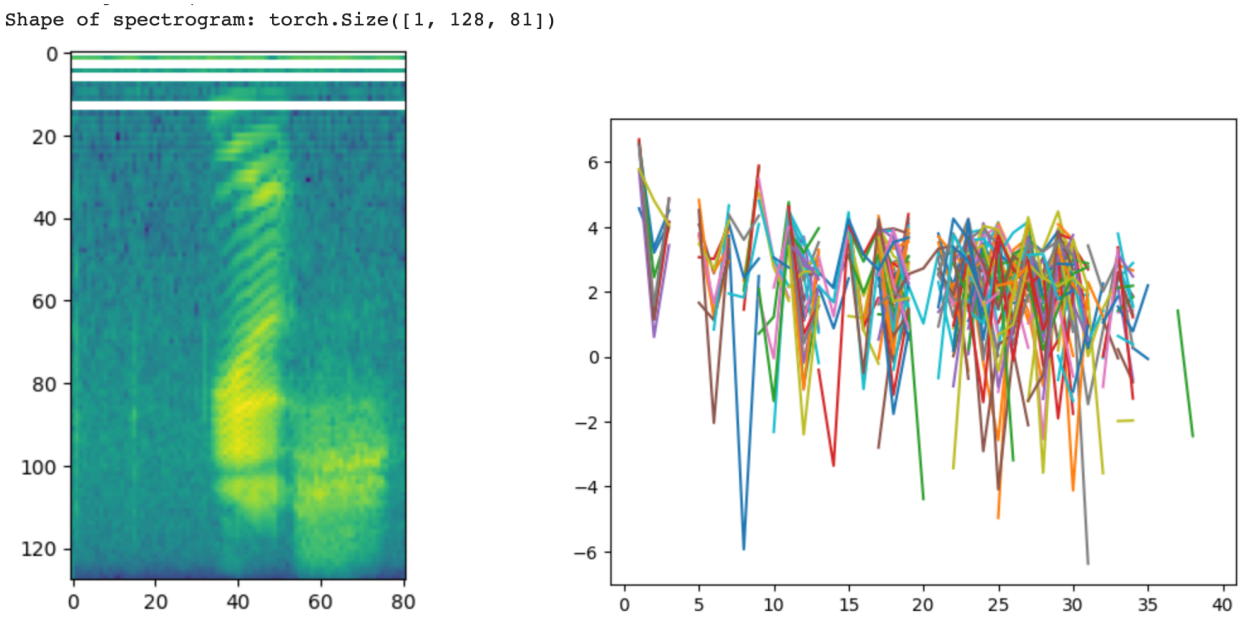
Spectrograms were employed to visualize the frequency components of audio signals. Using PyTorch's torchaudio.transforms, waveforms were transformed into spectrogram images. The resulting images served as input features for the convolutional neural network (CNN) model.

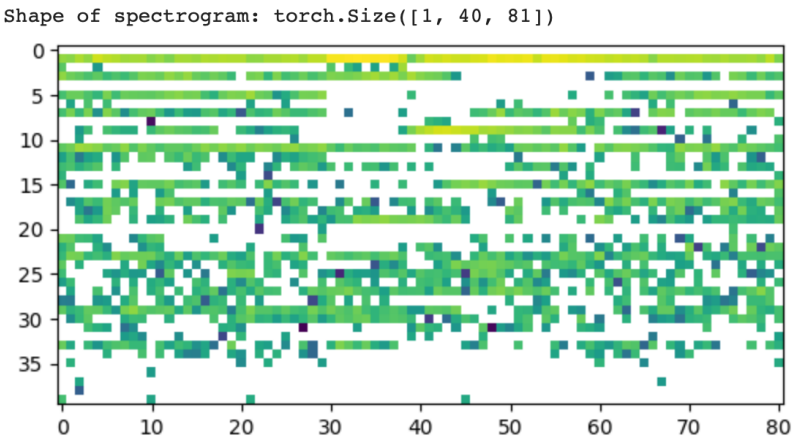
```
Shape of yes spectrogram: torch.Size([1, 201, 81])
Shape of no spectrogram: torch.Size([1, 201, 81])
```



Mel Spectrogram and MFCC

Additionally, Mel spectrograms and Mel-frequency cepstral coefficients (MFCC) were explored as alternative representations. Mel spectrograms provide a frequency-to-time mapping, and MFCC captures the amplitudes of the spectrum created from the frequency.





V. Model Architecture

The implemented model is a Convolutional Neural Network (CNN) designed to classify 'yes' and 'no' commands. The architecture includes convolutional layers, dropout layers, and fully connected layers. The final layer utilizes log softmax for classification.

VI. Challenges Encountered

1. Data Loading Issues:

During the implementation, some audio files were not found in the specified paths,

leading to FileNotFoundError and NotADirectoryError issues. This was addressed by cross verifying the file paths and ensuring the correct dataset was used.

2. Visualizing Spectrograms:

   Generating meaningful visualizations for spectrograms posed a challenge. Adjustments to parameters such as frame offsets and ensuring compatibility between waveform and spectrogram dimensions were required for accurate visualization.

VII. Model Deployment Consideration

The final CNN model, once functional, demonstrated satisfactory performance in classifying 'yes' and 'no' commands. Given the simplicity of the task and the availability of a well-prepared dataset, the model can be considered for deployment in scenarios requiring binary speech command classification.

VIII. Conclusion

This documentation summarizes the process of implementing an audio classification model using PyTorch based on the provided Microsoft tutorial. The dataset selection, data preparation, transformation, and visualization steps were outlined. Challenges encountered during implementation were addressed, resulting in a functional CNN model capable of classifying 'yes' and 'no' speech commands.

The final notebook implementing the model can be found in the assignment submission.