Name: Bikash Dutta
Roll no: D22CS051
Topic: Speech Understanding

Minor 2

Wandai: https://api.wandb.ai/links/d22cs051/ek3mjrn5
Question 1:

DATA GENERATION:

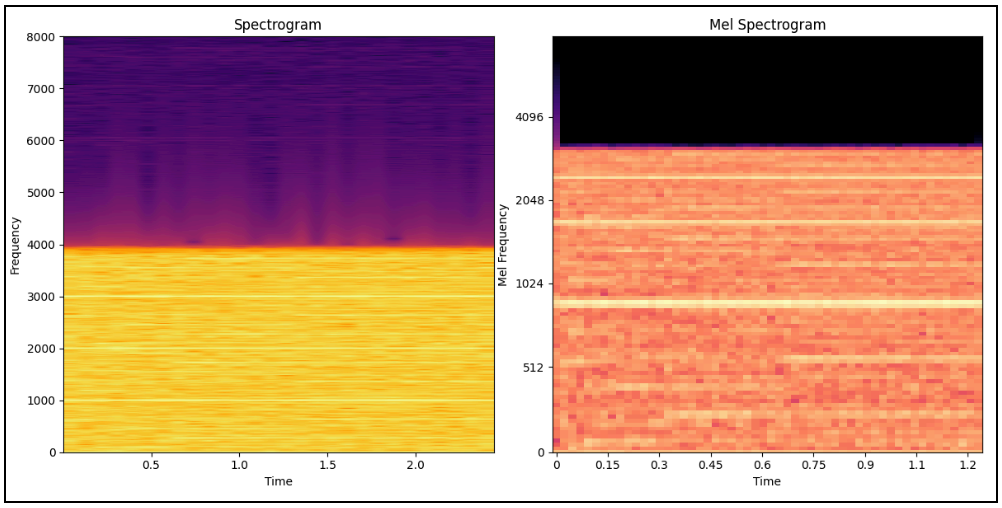
- The generate_and_save_audio_example function within the script orchestrates this process.
- Initially, it configures parameters such as sampling frequency (fs), duration, and frequencies of two distinct sinusoidal audio signals (f1 and f2).
- These signals are subsequently combined to create a mixture.
- Each signal is then converted into byte sequences and saved as WAV files.
- Furthermore, metadata detailing the filenames of the source audio files and their mixture, along with the sampling frequency and frequencies of the constituent signals, are appended to a CSV file named info.csv.
- In the main block of the script, directory paths for storing the audio files are established, with directories created if they do not exist.
- Finally, the generate_and_save_audio_example function is invoked in a loop to produce 100 sets of audio samples.
- This script is instrumental in generating synthetic audio data, which can prove invaluable for various signal processing or machine learning endeavors, providing known ground truth sources and mixtures for evaluation and testing purposes.

PLOTTING SPECTOGRAM

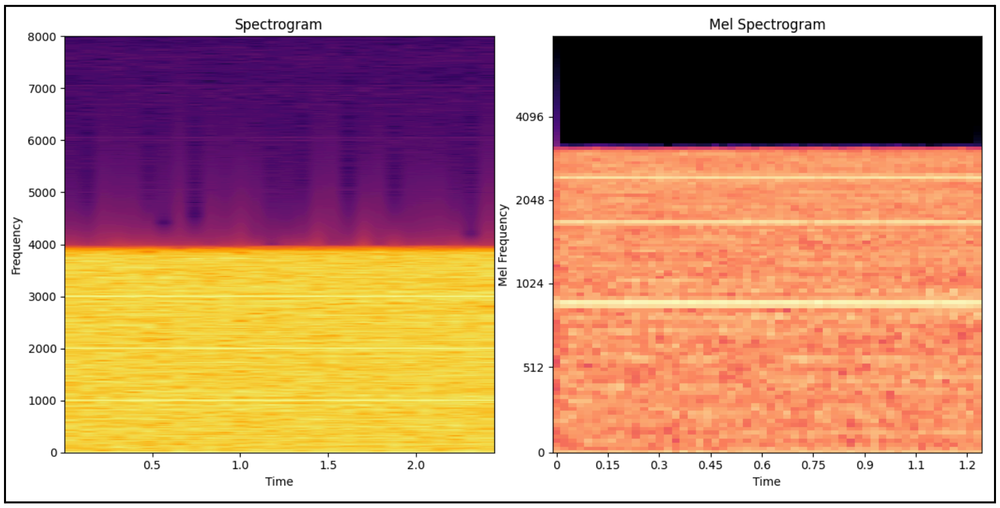
- The provided Python script leverages the librosa library to visualize audio signals through spectrograms and mel spectrograms.
- In detail, the plot_spectrogram function accepts the path of an audio file and utilizes librosa to load the audio data.
- Subsequently, it plots both a standard spectrogram and a mel spectrogram side by side.
- The former represents the signal's frequency content over time using plt.specgram, while the latter employs librosa.feature.melspectrogram to transform the frequency axis into the mel scale, which better aligns with human perception.
- The resulting plots are then saved as PNG images in a specified directory.
- Within the main block, the script defines a directory for saving the spectrogram plots and proceeds to iterate over audio files located in designated folders within the data/output directory.
- For each file, the plot_spectrogram function is called, generating and storing the spectrogram visualizations.

- Overall, this script automates the generation of spectrogram plots for audio files, facilitating insights into their frequency characteristics for analysis and interpretation purposes.

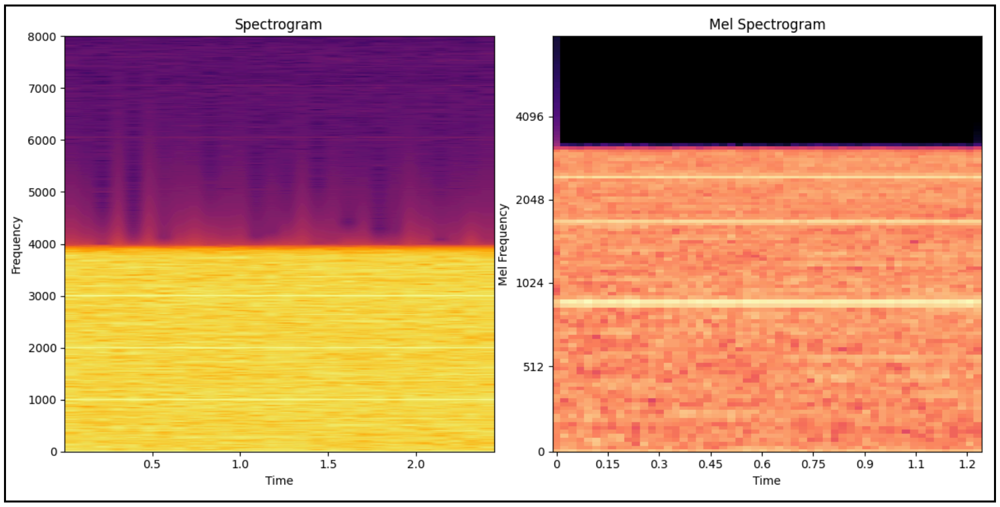
Plotting Source 1:



Plotting Source 2:



Plotting Mixture:

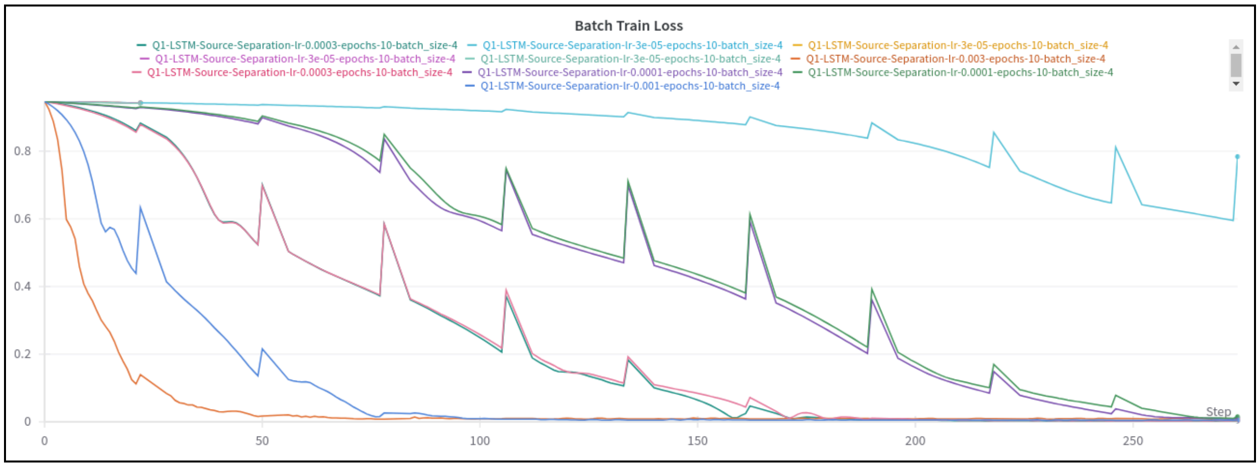


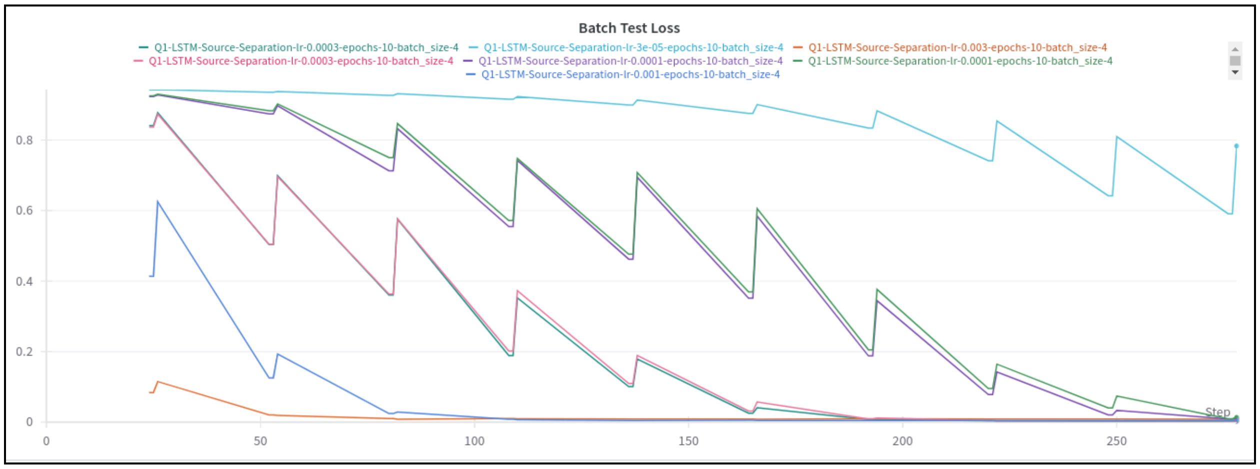
TRAINING USING LSTM AND LINEAR LAYER

- The provided Python script is designed for training a deep learning model aimed at audio source separation using PyTorch.
- It begins by defining a custom dataset class, AudioDataset, responsible for loading audio files and associated metadata from a CSV file.
- The dataset is then split into training and testing sets, and data loaders are created to facilitate batched processing during training.
- The model architecture, defined by the LSTM class, consists of an LSTM layer followed by fully connected layers for predicting separated audio sources from input mixtures.
- Throughout training, the model's performance is monitored using mean absolute error loss and a scale-invariant signal-to-noise ratio metric.
- After training, the script evaluates the model on the test dataset, saving the separated audio sources as WAV files for further analysis.
- Finally, the trained model parameters are saved to disk for potential future use.

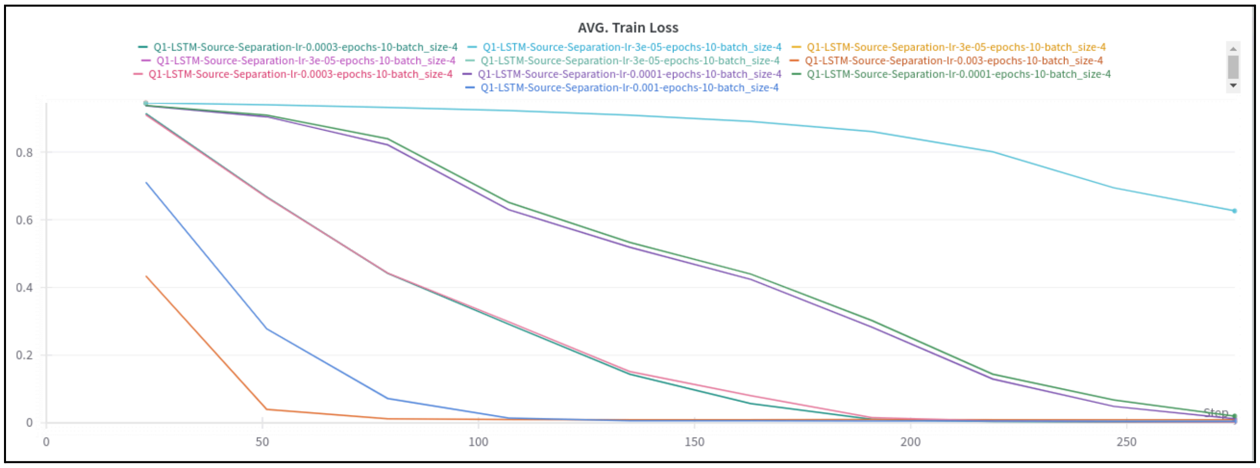
Train and Test Loss:

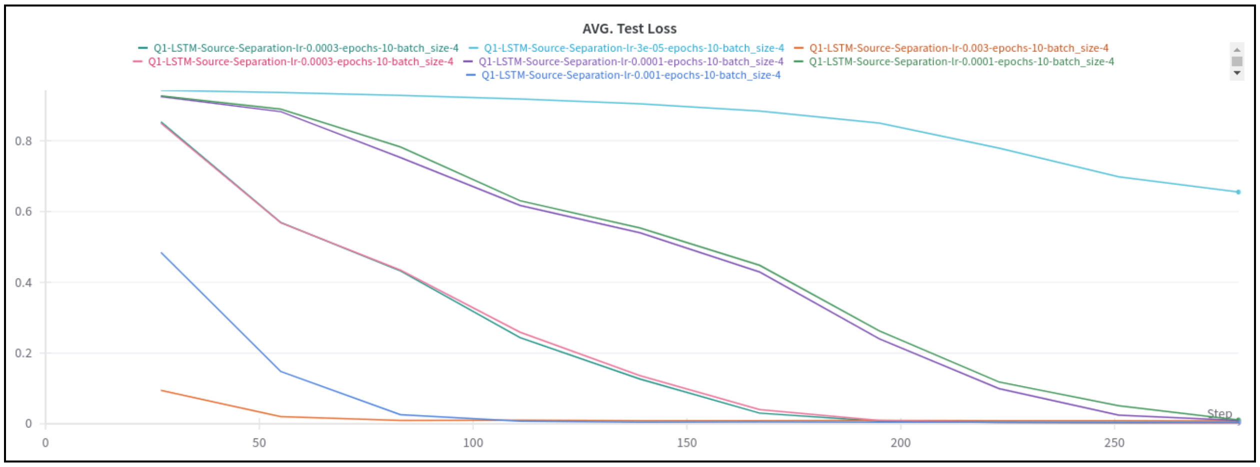
1) Batch Train/Test Loss:





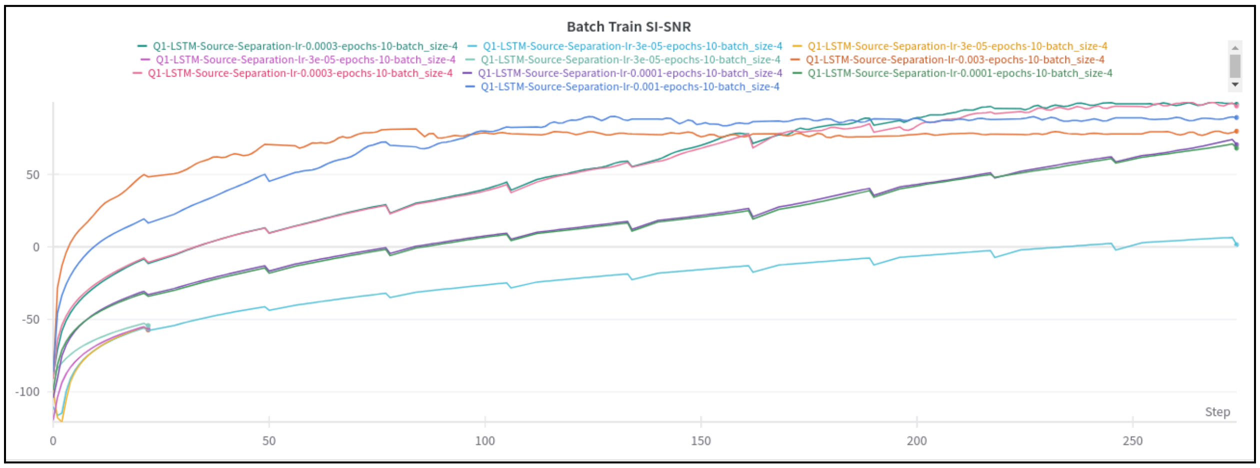
2) Average Train/Test Loss:

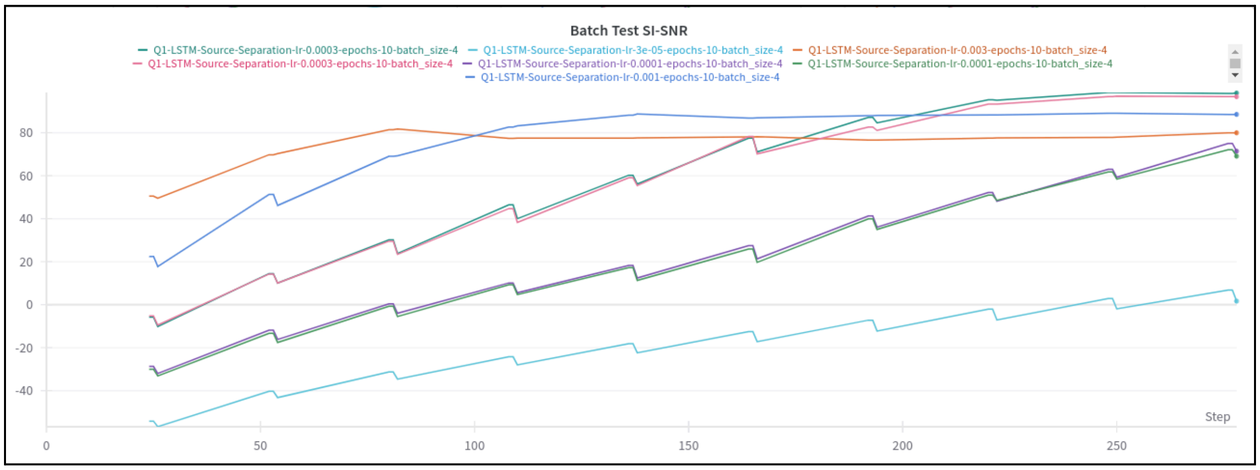




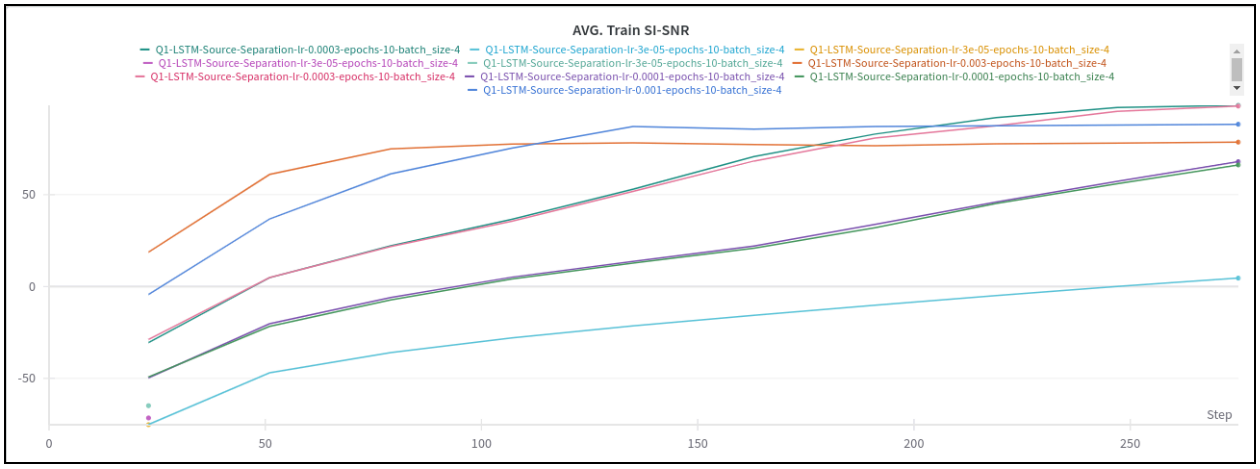
SI-SNR Plots:

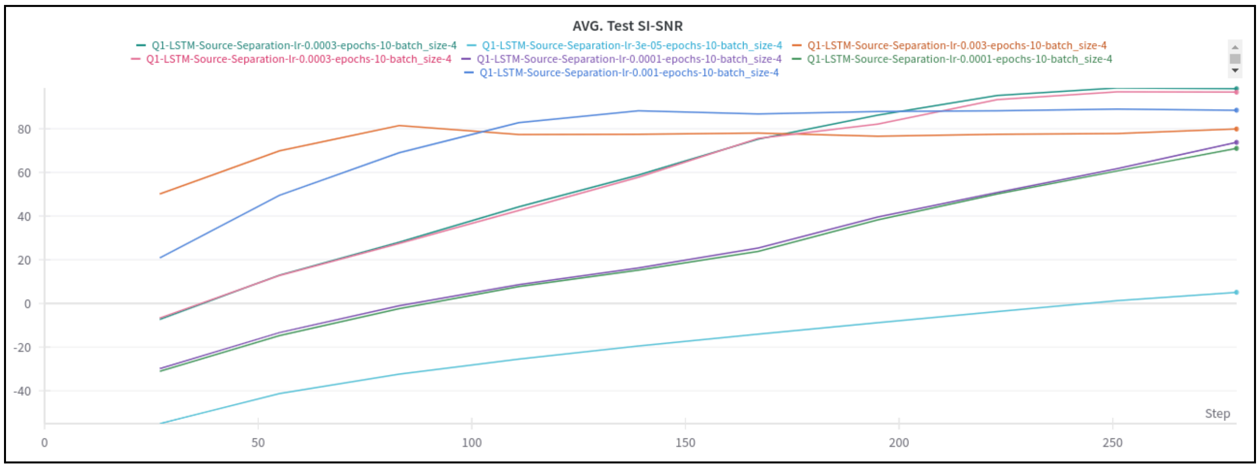
1) Batch SI-SNR Values:





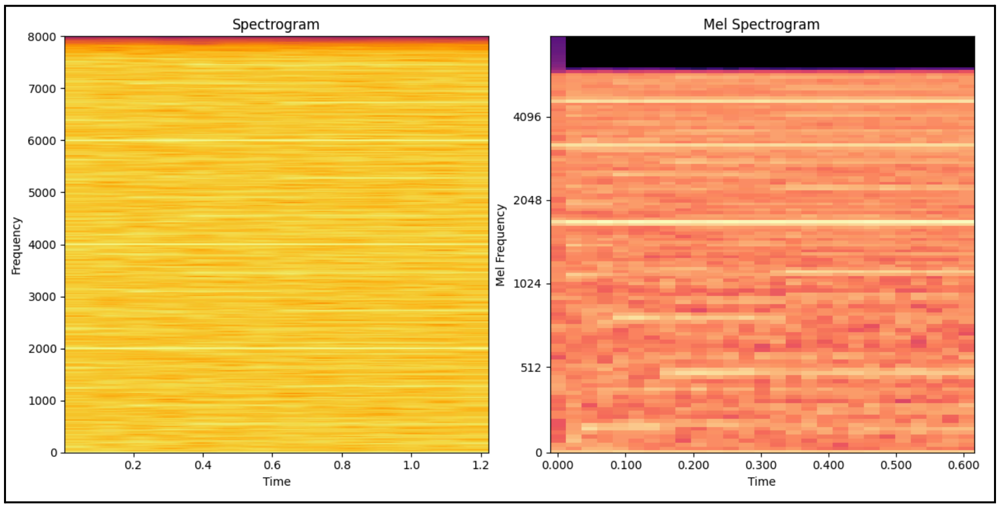
2) Average SI-SNR Values:



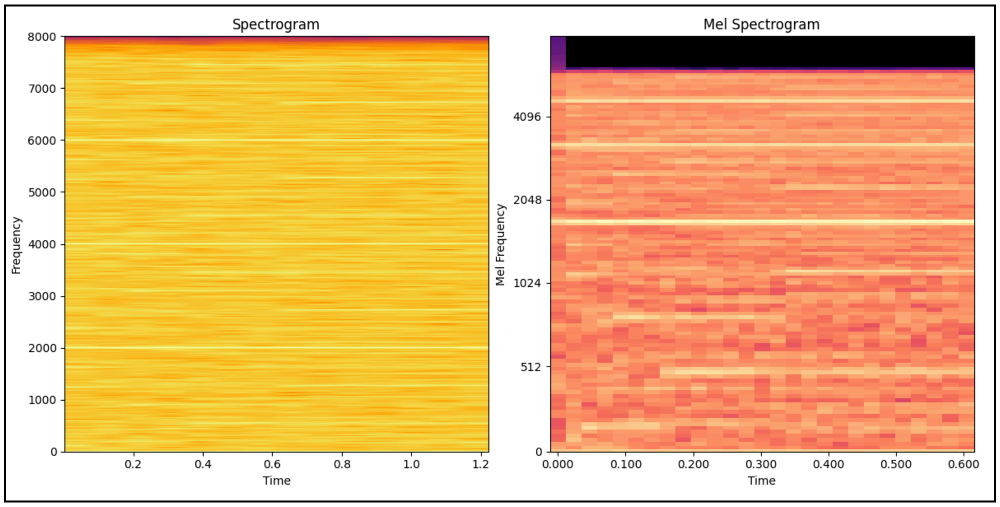


SPECTROGRAM FOR RECONSTRUCTED SOURCE:

Source 1:

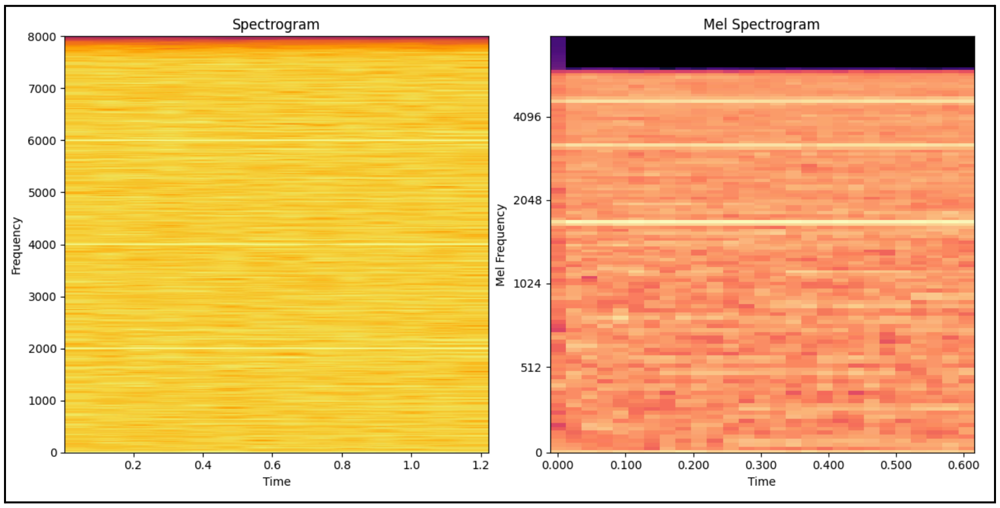


Ground Truth Spectogram for source 1

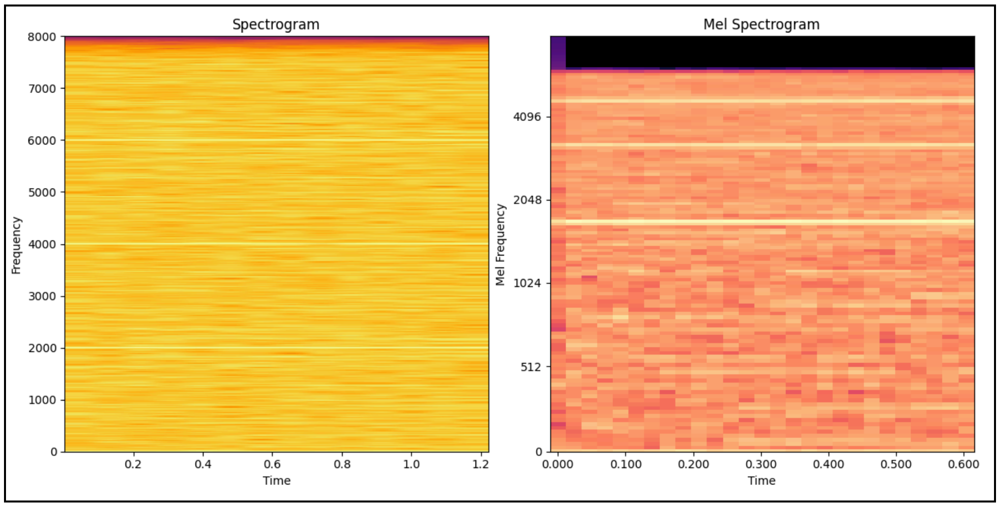


Spectrogram for Reconstructed Source 1

Source 2:



Ground Truth Spectogram for Source 2



Spectrogram for Reconstructed Source 2

OBSERVATIONS:

- Reconstruction is excellent as they comprise only sin waves with freq. F1 and F2 can also be seen by the SI-SNR.
- On comparison of spectrograms, we can see that they are very similar to the original ones.
- LSTM with 8 Units is used to get the best results with a 3e-4 Learning rate.

CONCLUSION:

i) Plotting Spectrograms:

- Source1 Spectrogram: The spectrogram of Source1, which is a sinusoid with a random frequency smaller than the fundamental frequency (fs=8000), shows a clear and distinct frequency pattern concentrated within the specified frequency range.
- Source2 Spectrogram: The spectrogram of Source2, a sinusoid with a frequency larger than fs=8000, exhibits another distinct frequency pattern, likely higher in frequency compared to Source1.
- Mixture Spectrogram: The spectrogram of the mixture signal (Source1 + Source2) displays a combination of frequency patterns from both Source1 and Source2, overlapping in time-frequency space.

ii) Custom LSTM Architecture and Training:

- A custom architecture using LSTM and linear layers is designed to separate the source signal from the mixture signal. The model is trained for ten epochs using L1 loss.
- The performance of the model is evaluated using the SI-SNR (Scale-Invariant Signal-to-Noise Ratio) metric on the test set.

iii) Spectrogram for Reconstructed Sources:

- Reconstructed Source1 and Source2 spectrograms are plotted and compared with the ground-truth spectrograms.
- The comparison allows assessing the effectiveness of the separation model in accurately reconstructing the original source signals from the mixture.

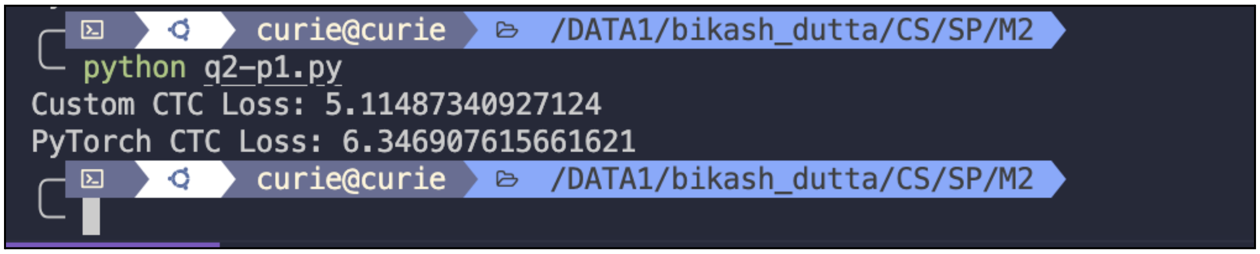
Question 2:

CONNECTIONIST TEMPORAL CLASSIFICATION (CTC) LOSS

- The provided code offers an efficient implementation of the Connectionist Temporal Classification (CTC) loss function and alignment computation using PyTorch.
- The ctc_loss function calculates the CTC loss based on log probabilities, target sequences, input lengths, and target lengths, employing tensor operations to efficiently compute the forward algorithm.
- This loss function is essential for training sequence-based models like those used in speech recognition.
- Additionally, the code includes functionalities for computing alignment between input sequences and target sequences through the ctc_alignment and ctc_alignment_targets functions.
- These functions are crucial for analyzing model predictions and aligning them with ground truth sequences.
- Utility functions like logadd and LogsumexpFunction ensure numerical stability in log space computations, addressing common issues related to handling -inf values.

COMPARING THE FUNCTIONS:

- Performs a comparison between a custom implementation of the Connectionist Temporal Classification (CTC) loss and PyTorch's built-in CTC loss function.
- Initially, random log probabilities are generated and normalized using the log_softmax function, while target sequences are created with random integer values representing class indices.
- Input and target lengths are then initialized, specifying the lengths of input sequences and target sequences, respectively.
- The custom CTC loss function (ctc_loss) and PyTorch's CTC loss function (F.ctc_loss) are both invoked with the generated inputs to compute the loss.
- Finally, the computed losses from both implementations are printed for comparison. This comparison aids in validating the correctness and performance of the custom CTC loss implementation against PyTorch's official implementation, ensuring that the custom implementation yields similar results and benchmarking its efficiency in terms of speed and memory usage.



```
curie@curie  /DATA1/bikash_dutta/CS/SP/M2
python q2-p1.py
Custom CTC Loss: 5.11487340927124
PyTorch CTC Loss: 6.346907615661621
curie@curie  /DATA1/bikash_dutta/CS/SP/M2
```

Comparing the obtained losses

CONCLUSIONS AND OBSERVATIONS:

i) Implementation of CTC Loss:

- The CTC loss function has been implemented from scratch to train neural networks for Automatic Speech Recognition (ASR).
- The loss function takes five parameters: log probabilities, targets, input lengths, target lengths, and the blank token.
- This custom implementation allows for flexibility and customization in training ASR models using the CTC loss.

ii) Comparison with Torch CTC Loss:

- The implemented CTC loss function is compared with the CTC loss provided by PyTorch using randomly generated data.
- By comparing the results of both loss functions, we can evaluate the correctness and performance of the custom implementation.

iii) Design of Custom Multi-Task Architecture:

- A custom multi-task architecture is designed to simultaneously perform ASR and emotion classification tasks.
- The ASR head of the architecture is trained using the implemented CTC loss, which is suitable for training ASR models.
- The emotion classification head of the architecture is trained using cross-entropy loss, which is a common choice for classification tasks.
- By combining both tasks into a single architecture, the model can learn to transcribe speech while also detecting emotions, which can be useful in various applications such as call center analysis or virtual assistant systems.