

Deep Learning (CSL4020)

Voice Controlled Music Recommendation System

Final Report

TEAM DESCRIPTION

Rishav Aich (B21AI029)

Pre-final Year (B.Tech. Artificial Intelligence & Data Science)

Ashutosh (B21AI007)

Pre-final Year (B.Tech. Artificial Intelligence & Data Science)

Saloni Garg (B21AI036)

Pre-final Year (B.Tech. Artificial Intelligence & Data Science)

Harsh Vardhan (B21BB012)

Pre-final Year (B.Tech. Bioscience & Bioengineering)

Problem Statement

“Design a deep learning model for Voice-controlled Music Recommendation System”

The goal is to develop a deep learning-based system that can understand a user's voice commands, analyze their music preferences, and provide personalized music recommendations. The key components include speech recognition, language understanding, user preference modeling, and music recommendation, all integrated into an end-to-end deep learning framework.

Dataset

Dataset	Description	Purpose
<i>Librespeech corpus Dataset</i>	The Librispeech dataset, SLR12, comprises English speech recordings in FLAC format, ensuring lossless quality preservation without compromising original audio data.	Used for speech-to-text conversion due to its diverse speech samples and language variety.
<i>Spotify million song Dataset</i>	This dataset contains song names, artists names, link to the song and lyrics. This dataset can be used for recommending songs, classifying or clustering songs.	Utilized for music recommendation due to its extensive collection of high-quality music tracks spanning various genres.

Solution and Implementation

Model 1: Automatic Speech Recognition Model (ASR)

Dataset	<i>Librespeech corpus Dataset</i>
Features	<ul style="list-style-type: none">• Audio : <i>English speech recordings (.flac)</i>• Transcript: <i>Transcript of the speech recordings</i>

Data Processing:

Objective:

- **Task-1:** *Transform audio data into spectrograms.*
- **Task-2:** *Implement data augmentation on the training data.*
- **Task-3:** *Transform the text data into numerical sequences.*

Task-1: Transform Audio Data Into Spectrograms

What is spectrogram ?

Spectrograms are visual representations of the spectrum of frequencies in a sound or other signal as they vary with time. A spectrogram is created by taking the Fourier Transform of a signal, which decomposes the signal into its constituent frequencies.

How it is significant to ASR model ?

Spectrograms are excellent for ASR models as they condense both frequency and temporal speech dynamics. They handle non-stationary signals effectively, capturing speech's changing frequency content over time.

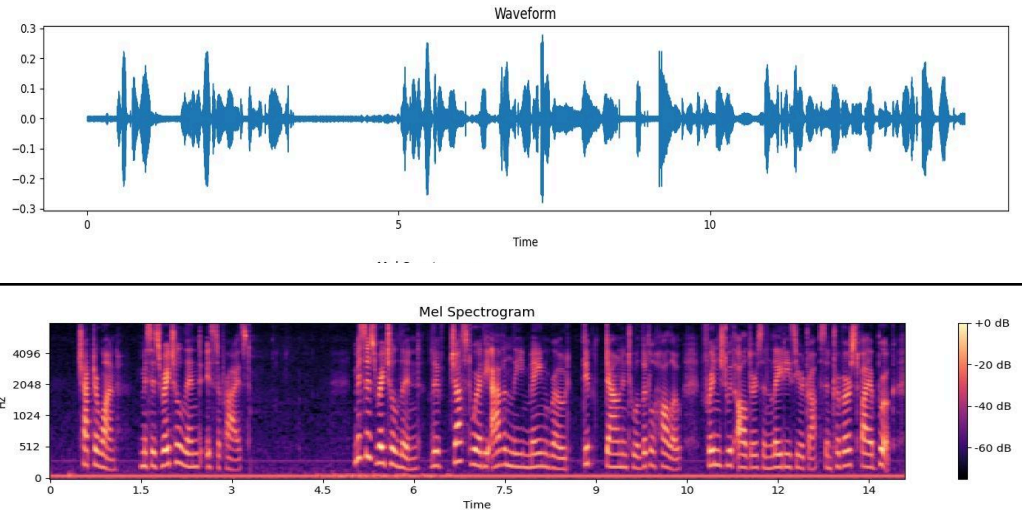
- The audio signal is transformed into a spectrogram using the Mel spectrogram.
- A mel spectrogram is a spectrogram where the frequencies are converted to the mel scale.
- Humans perceive frequencies non-linearly; lower frequencies have greater perceptual differences than higher frequencies.
- The mel scale transforms frequencies to a scale where equal pitch intervals sound equally distant to listeners, providing a more perceptually uniform representation.

$$S = \log(M(f))$$

M(f) is the power spectrum of the audio signal, and (S) is the spectrogram.

Speech Text

CHAPTER ONE MISSUS
RACHEL LYNDE IS
SURPRISED MISSUS
RACHEL LYNDE LIVED
JUST WHERE THE AVONLEA
MAIN ROAD DIPPED DOWN
INTO A LITTLE HOLLOW
FRINGED WITH ALDERS
AND LADIES EARDROPS
AND TRAVERSED BY A
BROOK



Task-2: Implement data augmentation on the training data

What is Data Augmentation ?

Data augmentation is a technique used in machine learning to artificially increase the size of the training dataset by creating modified versions of the existing data. This is particularly useful in tasks like automatic speech recognition (ASR), where the amount of high-quality, labeled data is often limited. By augmenting the data, models can learn from a wider variety of examples, potentially improving their generalization ability and performance on unseen data.

Augmentation Methods used on Training Data:

- **Frequency Masking :**
 - FrequencyMasking randomly masks a range of frequencies in the spectrogram.
 - This means that a certain range of frequencies in the audio signal is set to zero (or a specified value) for a certain number of frames.
 - The idea behind this is to simulate the effect of noise or other interferences that might occur in real-world audio signals.
 - By training the model on data with masked frequencies, the model learns to recognize speech even in the presence of such noise.

Mathematically,

$$S' = S \odot (1 - F)$$

(S) is the spectrogram, and (F) is the frequency mask, the operation can be represented as , where \odot denotes element-wise multiplication.

This operation effectively masks the frequencies in the spectrogram, making the model more robust to variations in frequency content.

- **Time Masking :**

- TimeMasking is similar to FrequencyMasking but operates on the time dimension of the spectrogram.
- It randomly masks a range of time frames in the spectrogram.
- This can simulate the effect of missing or distorted parts of the audio signal, such as those that might occur due to recording errors or background noise.
- By training the model on data with masked time frames, the model learns to recognize speech even when parts of the signal are missing or distorted.

Mathematically,

$$S' = S \odot (1 - T)$$

(S) is the spectrogram, and (T) is the frequency mask, the operation can be represented as , where \odot denotes element-wise multiplication.

This operation effectively masks the time frames in the spectrogram, making the model more robust to variations in the temporal structure of the audio signal.

Significance in ASR Model:

Both FrequencyMasking and TimeMasking are significant in ASR for several reasons:

- ❖ **Robustness:** By introducing variability into the training data, these techniques help the model become more robust to real-world conditions, such as noise, background sounds, and variations in the speaker's voice.
- ❖ **Generalization:** The model learns to recognize speech in a wider range of conditions, which can improve its performance on unseen data.
- ❖ **Efficiency:** Data augmentation techniques like these can be computationally efficient, as they do not require the collection of new data. They simply modify the existing data in various ways to create new training examples.

Task-3: Transform the text data into numerical sequences

This is a custom utility designed to facilitate the conversion between text and numerical sequences, which is a crucial step in preparing data for training and evaluating automatic speech recognition (ASR) models.

Key Components:

- **Character Mapping (*char_map*):**

- This dictionary maps each character in the English alphabet, including the space character, to a unique integer.
- The space character is mapped to a special token **<SPACE>** to distinguish it from other characters.
- This mapping is essential for converting text into numerical sequences, where each character is represented by its corresponding integer.

Method:

1. **text_to_int(text)**: This method takes a string of text as input and converts it into a sequence of integers.
2. It iterates over each character in the text, looks up the corresponding integer in **char_map**, and appends it to a list.
3. If the character is a space, it specifically looks up the **<SPACE>** token. The method returns the list of integers representing the text.

- **Index Mapping (*index_map*):**

- This dictionary serves as the inverse of **char_map**, mapping integers back to their corresponding characters.
- This is used to convert numerical sequences back into text, which is useful for decoding the output of the ASR model.

Method:

1. **int_to_text(labels)**: This method does the opposite of **text_to_int**.
2. It takes a sequence of integers (labels) as input and converts it back into a string of text.
3. It iterates over each integer in the sequence, looks up the corresponding character in **index_map**, and appends it to a list.
4. The method then joins the list of characters into a string and replaces the **<SPACE>** token with an actual space character. The method returns the decoded text.

Significance in ASR Model:

- ❖ This conversion process is crucial because ASR models typically work with numerical data, not raw text.
- ❖ By mapping text to integers and vice versa, the **Text_Transform** class enables the ASR system to effectively process and interpret speech data, ultimately leading to improved performance in speech recognition tasks.

Model Description And Performance Evaluation

<i>Model-1</i>	CNN + LSTM
<i>Model-2</i>	CNN + Bidirectional GRU
<i>Final Model</i>	CNN + ResNet + Bidirectional GRU

Before jumping onto the models lets understand the loss function, optimizer and evaluation metrics used.....

Loss Function

- The **Connectionist Temporal Classification (CTC)** loss function is used which is crucial for **Automatic Speech Recognition (ASR)** models, addressing the challenge of aligning variable-length input sequences (e.g., audio spectrograms) with output sequences (e.g., transcribed text).
- **CTC** allows the model to output sequences at any time step, without requiring alignment with the input sequence, making it particularly beneficial for **ASR**.

The mathematical expression for the CTC loss is:

$$L = -\frac{1}{N} \sum_{n=1}^N \log \left(\sum_{y \in Y} p(y|x_n) \right)$$

Where:

- L represents the CTC loss,
- N is the total number of samples,
- n represents the index of each sample,
- Y is the set of all possible label sequences,
- $p(y|x_n)$ is the probability of label sequence y given input x_n , and
- \log denotes the natural logarithm.

- **CTC's** significance lies in its ability to handle variable-length sequences, providing flexibility in alignment, computational efficiency, and robustness against errors in the input data, making it a powerful tool for **ASR** systems.

Optimizer

- The **AdamW optimizer**, an extension of the Adam optimizer, is particularly effective for training deep learning models such as ASR systems.

- It adapts the learning rate for each parameter, improving convergence and efficiently handling sparse gradients.

The update rule for Adam can be represented mathematically as follows:

$$\begin{aligned}
 m_t &= \beta_1 m_{t-1} + (1 - \beta_1) \nabla_{\theta} J(\theta) \\
 v_t &= \beta_2 v_{t-1} + (1 - \beta_2) (\nabla_{\theta} J(\theta))^2 \\
 \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\
 \hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \\
 \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t
 \end{aligned}$$

Where:

- m_t and v_t are moving averages of the gradients and squared gradients, respectively,
- \hat{m}_t and \hat{v}_t are bias-corrected estimates of the moving averages,
- η is the learning rate,
- ϵ is a small constant to prevent division by zero,
- t is the current time step,
- β_1 and β_2 are hyperparameters that control the decay rates of the moving averages.

- **AdamW's** update rule involves maintaining moving averages of past gradients and squared gradients, which are then used to adjust the learning rate.
- This method is robust, less sensitive to the initial learning rate, and computationally efficient, making it ideal for ASR tasks dealing with high-dimensional input data.

Evaluation Metrics

- **Levenshtein Distance:**
 - The Levenshtein distance is a measure of the difference between two sequences, defined as the minimum number of single-character edits (**insertions, deletions, or substitutions**) required to change one sequence into the other.

The proper mathematical expression for the Levenshtein distance d between two sequences s and t of lengths m and n is defined as:

$$d = \min \begin{pmatrix} d(i-1, j) + 1 & \text{if } s_i \neq t_j \\ d(i, j-1) + 1 & \text{if } s_i \neq t_j \\ d(i-1, j-1) & \text{if } s_i = t_j \end{pmatrix}$$

for all $1 \leq i \leq m$ and $1 \leq j \leq n$.

Significance:

- ❖ The **Levenshtein distance** is fundamental to the calculation of **WER** and **CER** because it quantifies the difference between the reference and hypothesis sequences, which is the basis for these metrics.

- ❖ It is a widely used measure in natural language processing and computational linguistics for comparing sequences of characters or words.

- **Character Error Rate (CER):**

- CER is a measure of the performance of an ASR system in terms of the number of character errors per sentence.
- It is calculated as the ratio of the number of substitutions, insertions, and deletions to the total number of characters in the reference transcription.

Mathematically,

$$CER = (Substitutions + Insertions + Deletions) / (Total Reference Characters)$$

Process:

1. The **char_errors** function calculates the **Levenshtein distance** between the reference and hypothesis sentences, considering them as sequences of characters.
2. This is similar to the word-level calculation but operates at the character level.
3. The **cer** function then uses the Levenshtein distance to calculate the **CER** by dividing the number of edits by the total number of characters in the reference sentence.

Significance:

- ❖ CER is significant because it provides a measure of the ASR system's performance at the character level, which can be particularly useful for languages where the number of characters is relatively small, such as English.
- ❖ It offers a more granular view of the ASR system's performance compared to WER.

- **Word Error Rate (WER):**

- WER is a measure of the performance of an ASR system in terms of the number of word errors per sentence.
- It is calculated as the ratio of the number of substitutions, insertions, and deletions to the total number of words in the reference transcription.

Mathematically,

$$WER = (Substitutions + Insertions + Deletions) / (Total Reference Words)$$

Process:

1. The **word_errors** function computes the **Levenshtein distance** between the reference and hypothesis sentences, treating them as sequences of words.

2. This distance represents the minimum number of edits (**insertions, deletions, or substitutions**) needed to transform one sequence into the other.
3. The wer function then calculates the **Word Error Rate (WER)** by dividing the **Levenshtein distance** by the total number of words in the reference sentence, providing a measure of word-level transcription accuracy.

Significance:

- ❖ WER is significant because it directly measures the accuracy of the ASR system in terms of word-level transcription.
- ❖ It is widely used in ASR research and industry to evaluate the performance of speech recognition systems.

Model-1: CNN + LSTM

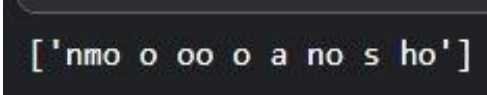
The first model combines a Convolutional Neural Network (CNN) with a Long Short-Term Memory (LSTM) network.

Strategy and Analysis:

- **CNNs**
 - Are excellent at extracting local features from spectrograms, which are essentially 2D representations of audio signals.
 - By applying convolutional layers, CNNs can learn to detect patterns in the frequency and time domains of the audio signal.
 - This makes them particularly effective at feature extraction for ASR tasks, where the goal is to convert audio signals into text.
- **LSTMs**
 - Are designed to capture the temporal dependencies in sequences of data.
 - After the CNN has extracted features from the spectrograms, the LSTM processes these features over time to capture the temporal dynamics of the speech.
 - This is crucial for understanding the sequence of words and the context in which they are spoken.
- **CNN + LSTM**
 - In an ASR system, the CNN is used to extract features from the spectrograms, and the LSTM is used to process these features over time.
 - This combination allows the model to learn both local features from the spectrograms and the temporal dependencies in the sequence of features, which is essential for accurately transcribing speech.

Results

Average Loss	3.1684
Average CER	1.00
Average WER	0.8335

Actual	Predicted
<i>Hello suggest me a sad song</i>	

Proposed Reason for poor results:

- **Lack of Residual Connections:**
 - The **LSTM** does not inherently include residual connections, which are a key feature of **ResNet** architectures.
 - Residual connections help in mitigating the vanishing gradient problem, allowing the model to learn more complex patterns by providing shortcuts for gradients to flow through the network.
- **LSTM Complexity:**
 - **LSTMs** are more computationally intensive and complex than **GRUs (Gated Recurrent Units)**, which might lead to longer training times and potentially overfitting if not properly regularized.

Model-2: CNN + Bidirectional GRU

The second model adds a Bidirectional GRU to the CNN.

Strategy and Analysis:

- **Bidirectional GRU**
 - In the context of Automatic Speech Recognition (ASR), the ability to capture both past and future context is crucial for accurately transcribing speech.
 - Speech is a sequential process where the meaning of a word can depend on both its preceding and following words.

- For example, the word "no" can mean "not" when it precedes a verb, but it can mean "no" when it follows a negative adjective.
- By processing the input sequence in both directions, a Bidirectional GRU can capture this context, allowing the model to make more accurate predictions.
- This is particularly important in ASR, where the goal is to transcribe spoken language into written text. Accurately capturing the context of each word is essential for producing correct transcriptions.

● CNN + Bidirectional GRU

- When combined with a Convolutional Neural Network (CNN), the Bidirectional GRU can process both the spatial features extracted by the CNN and the temporal context captured by the GRU.
- The CNN is responsible for extracting features from the input spectrograms, which are then fed into the GRU.
- The GRU processes these features in both forward and backward directions, capturing the temporal context of the speech.
- This combination allows the model to effectively learn from both the spatial and temporal aspects of the speech data, potentially leading to better performance in ASR tasks.
- The CNN can learn to recognize patterns in the spectrograms, while the GRU can learn to understand the sequence of these patterns over time.

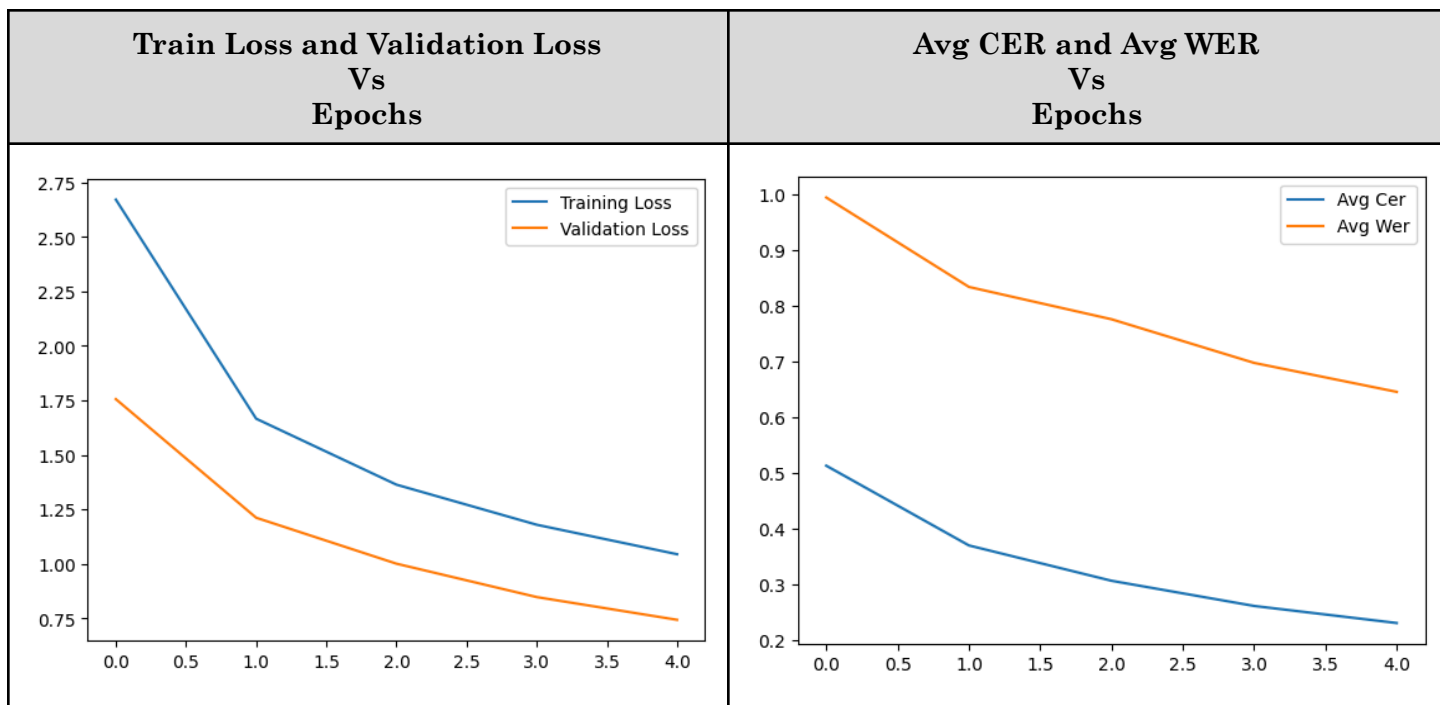
Results

```

-----
Train Epoch: 5 [0/25685 (0%)] Loss: 1.023636
Train Epoch: 5 [1000/25685 (4%)] Loss: 1.079134
Train Epoch: 5 [2000/25685 (8%)] Loss: 1.107488
Train Epoch: 5 [3000/25685 (12%)] Loss: 1.150121
Train Epoch: 5 [4000/25685 (16%)] Loss: 1.059502
Train Epoch: 5 [5000/25685 (19%)] Loss: 0.879048
Train Epoch: 5 [6000/25685 (23%)] Loss: 1.007533
Train Epoch: 5 [7000/25685 (27%)] Loss: 1.023941
Train Epoch: 5 [8000/25685 (31%)] Loss: 1.075178
Train Epoch: 5 [9000/25685 (35%)] Loss: 0.872702
Train Epoch: 5 [10000/25685 (39%)] Loss: 1.056794
Train Epoch: 5 [11000/25685 (43%)] Loss: 0.993934
Train Epoch: 5 [12000/25685 (47%)] Loss: 1.126958
Train Epoch: 5 [13000/25685 (51%)] Loss: 1.060097
Train Epoch: 5 [14000/25685 (54%)] Loss: 0.920996
Train Epoch: 5 [15000/25685 (58%)] Loss: 1.240468
Train Epoch: 5 [16000/25685 (62%)] Loss: 1.041801
Train Epoch: 5 [17000/25685 (66%)] Loss: 0.937524
Train Epoch: 5 [18000/25685 (70%)] Loss: 0.942698
Train Epoch: 5 [19000/25685 (74%)] Loss: 0.885349
Train Epoch: 5 [20000/25685 (78%)] Loss: 1.044407
Train Epoch: 5 [21000/25685 (82%)] Loss: 0.995170
Train Epoch: 5 [22000/25685 (86%)] Loss: 0.920371
Train Epoch: 5 [23000/25685 (90%)] Loss: 1.098228
Train Epoch: 5 [24000/25685 (93%)] Loss: 0.901584
Train Epoch: 5 [25000/25685 (97%)] Loss: 0.900328

evaluating...
Validation set: Average loss: 0.7437, Average CER: 0.231427 Average WER: 0.6465
-----

```



Average Loss	0.7437
Average CER	0.2314
Average WER	0.6465

Actual	Predicted
<i>Hello suggest me a sad song</i>	<code>['a ok sethis made no tedso']</code>

Test Results:

```

he hoped there would be stew for dinner turnips and carrots and bruised potatoes and fat mutton pieces to be ladled out in thick peppered flour fattened sauce
he hoped there would be stoom for dinner turnips and carit send brosed be taehos and fat mutn peces to be laele doutanfick pepered fouwer fetan ses
-----
stuff it into you his belly counselled him
stuffid into you his belly counsald him
-----
after early nightfall the yellow lamps would light up here and there the squalid quarter of the brothels
after erly night fall the en aw lamps wild lihte hop pear in there the squalled corter of the brofls
-----

```

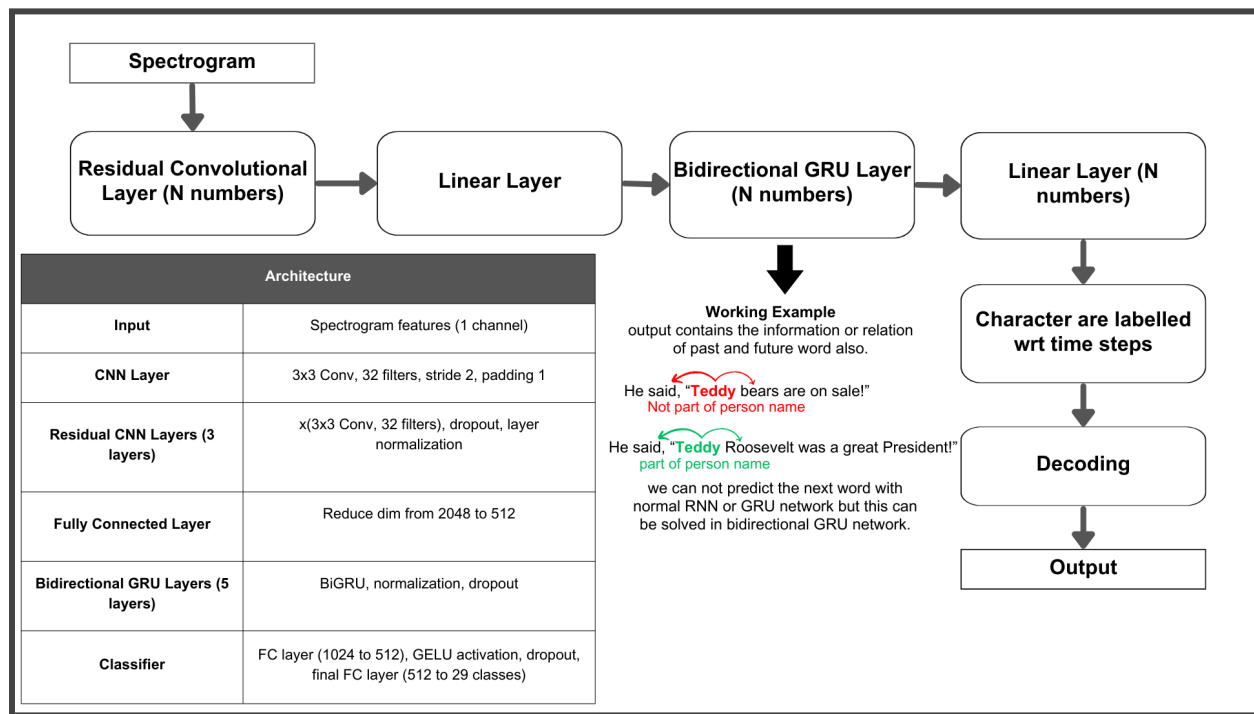
Proposed Reason for poor results:

- **Limited Depth of Residual Connections:**
 - While the addition of a GRU introduces some form of residual connections, it does not include the depth of connections found in ResNet architectures.

- This might limit the model's ability to learn complex patterns in the data.
- **GRU Complexity:**
 - GRUs are less complex than LSTMs but still more complex than simple feedforward networks.
 - This added complexity might not be fully justified by the performance gains, especially if the model is not properly tuned or if the data does not require the depth of context captured by bidirectional processing.

Model-3: CNN + ResNet + Bidirectional GRU

The third model combines a CNN with a ResNet architecture and a Bidirectional GRU. This model likely outperforms the previous two for several reasons:



Strategy and Analysis:

- **Residual Connections:**
 - The ResNet architecture includes residual connections, which help in learning complex patterns by allowing gradients to flow more easily through the network.
 - In ASR, where the model often consists of multiple layers to process the complexities of speech data, residual connections can significantly improve the learning process.
 - By allowing gradients to flow more easily through the network, the model can learn more complex patterns in the data, such as the intricate dependencies between different parts of a speech signal.

- This is particularly beneficial for tasks like ASR, where the meaning of a word can depend on its context, which can span multiple layers of the network.

- **Depth of Context:**

- ASR models require capturing a depth of context to accurately transcribe speech, understanding not just immediate surroundings but also broader context spanning words or sentences.
- The combination of a Bidirectional GRU with ResNet architecture enables this by processing input in both forward and backward directions, capturing both past and future context.
- This dual-direction processing is beneficial for speech, where word meaning depends on its preceding and following words, crucial for tasks like ASR.

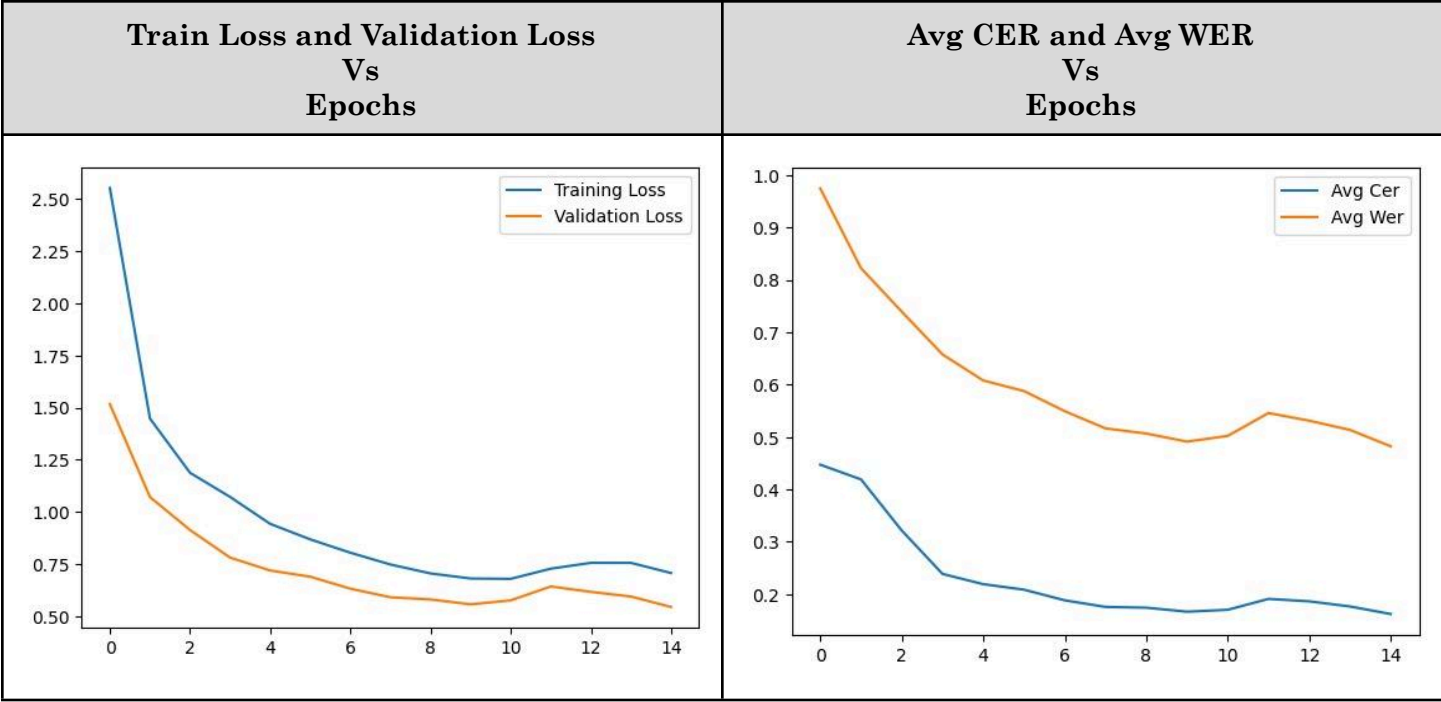
- **Efficiency:**

- **Parameter Sharing:** In ResNet, the same weights are used for the shortcut connections, which reduces the number of parameters compared to traditional architectures where each layer has its own set of weights.
- **Gradient Flow:** The residual connections help in maintaining a good flow of gradients through the network, which can lead to faster convergence and better performance with fewer parameters.
- **Reduced Overfitting:** By allowing the model to learn from the residual connections, ResNet can reduce overfitting, which is a common issue in deep networks. This can lead to better generalization to unseen data.

Results

```
-----
Train Epoch: 5 [0/23116 (0%)]    Loss: 1.004785
Train Epoch: 5 [1000/23116 (4%)]    Loss: 0.853835
Train Epoch: 5 [2000/23116 (9%)]    Loss: 1.083526
Train Epoch: 5 [3000/23116 (13%)]    Loss: 0.940021
Train Epoch: 5 [4000/23116 (17%)]    Loss: 0.948542
Train Epoch: 5 [5000/23116 (22%)]    Loss: 0.849128
Train Epoch: 5 [6000/23116 (26%)]    Loss: 0.896162
Train Epoch: 5 [7000/23116 (30%)]    Loss: 1.026917
Train Epoch: 5 [8000/23116 (35%)]    Loss: 0.986133
Train Epoch: 5 [9000/23116 (39%)]    Loss: 1.003344
Train Epoch: 5 [10000/23116 (43%)]    Loss: 0.911211
Train Epoch: 5 [11000/23116 (48%)]    Loss: 0.865564
Train Epoch: 5 [12000/23116 (52%)]    Loss: 0.822749
Train Epoch: 5 [13000/23116 (56%)]    Loss: 1.019057
Train Epoch: 5 [14000/23116 (61%)]    Loss: 0.864678
Train Epoch: 5 [15000/23116 (65%)]    Loss: 0.883692
Train Epoch: 5 [16000/23116 (69%)]    Loss: 0.874675
Train Epoch: 5 [17000/23116 (74%)]    Loss: 0.918670
Train Epoch: 5 [18000/23116 (78%)]    Loss: 0.981945
Train Epoch: 5 [19000/23116 (82%)]    Loss: 0.900044
Train Epoch: 5 [20000/23116 (87%)]    Loss: 1.092802
Train Epoch: 5 [21000/23116 (91%)]    Loss: 0.987747
Train Epoch: 5 [22000/23116 (95%)]    Loss: 0.851666
Train Epoch: 5 [23000/23116 (99%)]    Loss: 0.890449

evaluating...
Validation set: Average loss: 0.6562, Average CER: 0.203498 Average WER: 0.5787
```



Average Loss	0.6562
Average CER	0.2035
Average WER	0.5787

Actual	Predicted
Hello suggest me a sad song	['a lent sa wist me a saiso']

Test Result

he hoped there would be stew for dinner turnips and carrots and bruised potatoes and fat mutton pieces to be ladled out in thick peppered flour fattened sauce
he hoped there would be sto her dinner turnips and caratsand brosed petatos and tat buten pieaces to be latled outinth thic peppered flowr facan saus

stuff it into you his belly counselled him
stuffid in to you his belly counstiled him

after early nightfall the yellow lamps would light up here and there the squalid quarter of the brothels
after early night fall the yelnlow lamps would light hap phere ind there the squalled quarter of the brofls

Summarizing the Comparison between the models

	CNN + Bidirectional GRU	CNN + ResNet + Bidirectional GRU
Average Loss	0.7437	0.6562
Average CER	0.2314	0.2035
Average WER	0.6465	0.5787
Result	Hello suggest me a sad song	
	['a ok sethis made no tedso']	['a lent sa wist me a saiso']

Model 2: Music Recommendation Model from Speech Text

Dataset	Spotify million song dataset
Features	<ul style="list-style-type: none">• Artist name : Name of the singer• Song Name: Title of the song• Text: Lyrics of the song

For recommending music from the speech text we use two methods:

1. String matching
2. Embedding-Based Recommendation

Fuzzy String Matching

In this method we are matching the speech query with the artist name and song title / song name.

- Fuzzy string matching is a technique used to find strings that are approximately equal to a given pattern.
- The **FuzzyWuzzy** library in Python provides a simple and effective way to perform fuzzy string matching using the Levenshtein distance, which measures the minimum number of single-character edits (**insertions, deletions, or substitutions**) required to change one word into the other.
- For each dataset entry, the **Levenshtein distance** is calculated, and a similarity score is assigned. The similarity score is a measure of how similar the dataset entry is to the query. The lower the score, the more similar the entry is to the query.

The Levenshtein distance between two strings s and t is defined as:

$$d(s, t) = \min \left(\begin{array}{ll} d(s, t) & \text{if } s = \emptyset \text{ or } t = \emptyset \\ d(s, t) & \text{if } s = t \\ 1 + d(s, t) & \text{if } s \neq t \\ 1 + \min \left(\begin{array}{ll} d(s_1, t) & \text{if } s_1 \neq t_1 \\ d(s, t_1) & \text{if } s \neq t_1 \\ d(s_1, t_1) & \text{if } s_1 \neq t_1 \text{ and } s \neq t_1 \end{array} \right) & \text{otherwise} \end{array} \right)$$

Where s and t are the two strings being compared, and $d(s, t)$ is the Levenshtein distance between them.

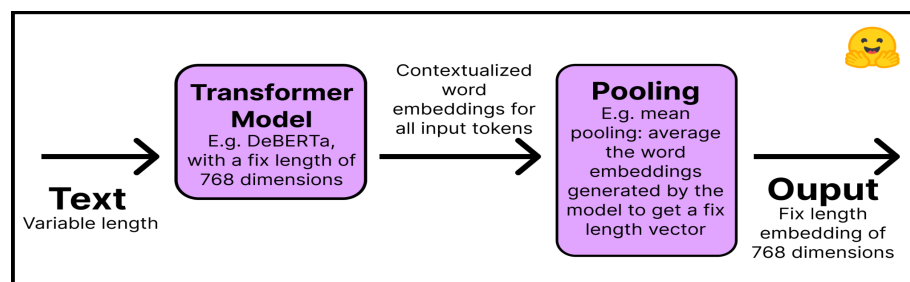
Advantages:

- ❖ **Handling Typos and Variations:** Fuzzy string matching is particularly useful for handling queries that may not exactly match the entries in the dataset, such as misspellings or variations in wording.
- ❖ **Straightforward Implementation:** It provides a straightforward way to recommend items based on a similarity score, making it easy to understand and implement.
- ❖ **Flexibility:** The method can be easily adjusted to prioritize different types of edits (insertions, deletions, substitutions) by modifying the scoring function.

Embedding-Based Recommendation

Step 1: Text to Embedding Conversion

- **Algorithm:** SentenceTransformer model
- **Process:**
 - The query text is passed through the SentenceTransformer model, which is a transformer-based model designed for generating sentence embeddings.
 - These embeddings are high-dimensional vectors that represent the semantic meaning of the input text.

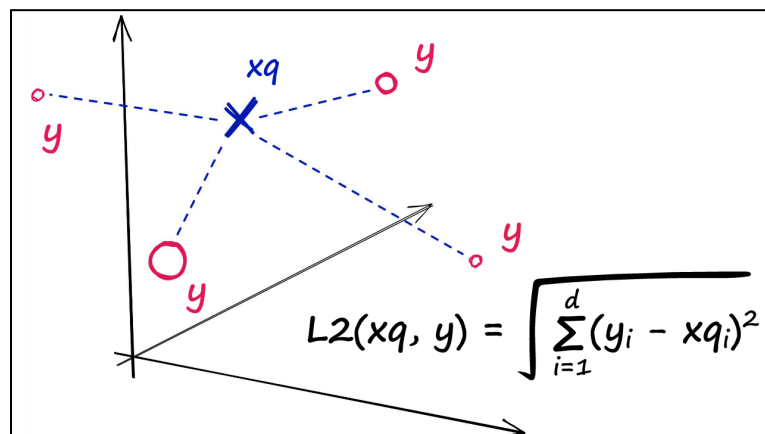


- **Architecture:**
 - The SentenceTransformer model uses transformer architecture, which involves self-attention mechanisms and positional encoding.

- The output embedding is a vector in a high-dimensional space, where the distance between vectors can be interpreted as the **semantic similarity** between the texts they represent.

Step 2: Indexing Embeddings

- **Algorithm:** Faiss (Facebook AI Similarity Search)
- **Process:**
 - The embeddings of the dataset are indexed using Faiss.
 - **Faiss** is a library developed by **Facebook AI Research (FAIR)** that provides efficient similarity search and clustering of dense vectors.
 - It allows for the creation of an index that can be searched to find the nearest neighbors to a given query embedding.
- **Mathematical Representation:**
 - The embeddings are represented as vectors in a high-dimensional space.
 - The distance between vectors is calculated using a distance metric, such as **Euclidean distance or cosine similarity**.



L2 distance calculation between a query vector xq and our indexed vectors (shown as y)

- In the context of Faiss, the index is built using a flat index (**e.g., IndexFlatL2 for Euclidean distance**) or an **IVF (Inverted File)** index for more efficient search.
- **IndexFlatL2** measures the **L2 (or Euclidean) distance** between all given points between our query vector, and the vectors loaded into the index. It's simple, very accurate, but not too fast.

Step 3: Nearest Neighbor Search

- **Algorithm:** Faiss Index Search
- **Process:**
 - The query embedding is searched against the Faiss index to find the nearest neighbors.
 - This involves calculating the distance between the query embedding and each embedding in the index and returning the embeddings that are closest to the query.

- **Mathematical Representation:**

Mathematically, the distance (d) between two vectors (\mathbf{v}_1 and \mathbf{v}_2) using the Euclidean distance metric is calculated as:

$$d = \sqrt{\sum_{i=1}^n (v_{1i} - v_{2i})^2}$$

Where:

- n is the dimension of the vectors,
- v_{1i} and v_{2i} are the components of vectors \mathbf{v}_1 and \mathbf{v}_2 respectively.

Step 4: Aggregating Results

- **Process:**

- The system aggregates the results from the nearest neighbor search to provide a comprehensive recommendation.
- This involves selecting the most relevant results based on the similarity scores and possibly applying additional logic to ensure diversity in the recommendations.

The relevance of each result is determined by its **similarity score**, which is a measure of how similar the result's embedding is to the query embedding. Higher similarity scores indicate more relevant results.

Advantages:

- ❖ **Semantic Understanding:** By converting text into embeddings, the system can capture the semantic meaning of the text, making it suitable for tasks like music recommendation where the context and meaning of the query are important.
- ❖ **Flexibility:** The method can handle a wide range of queries and is not limited to exact matches. It can find relevant results even when the query does not exactly match any entry in the dataset.

Combining Both The Methods

- In the provided code, both methods are used in conjunction to recommend music artists and songs.
- The embedding-based recommendation serves as the primary method, providing a comprehensive recommendation based on the semantic meaning of the query.
- The fuzzy string matching method acts as a fallback mechanism, ensuring that the system can still provide recommendations even when the embedding-based search does not yield satisfactory results.
- This combination allows the system to handle a wide range of queries and provide accurate recommendations.

Results

Actual Query	Prediction
Bang	nbhang
Recommendations	
Closest Artist Recommendation : Zac Brown Band Songs Recommended : ['Mystery Babylon', 'Nocturnal Pleasure', 'The Villagers', 'Daffodils'] Top Song Recommended : A!	

Actual Query	Prediction
Suggest Me a Sad Song	a lent sa wist me a saiso
Recommendations	
Closest Artist Recommendation : O.A.R. Songs Recommended : ['Raat Barasaat Ki, Pahali Mulaqaat Ki', 'Nu Ma Las De Limba Noastra', 'Malia', 'Sore Tugu Pancoran'] Top Song Recommended : Bang-A-Boomerang	

Weaknesses Of The Models

- **Model 1: ASR Component**
 - **Accent Differences**
 - **Issue:** The model's predictions are highly accurate for the LibriSpeech dataset, which is a large corpus of English speech. However, it struggles with data outside this dataset, particularly when the accents differ significantly from those in the LibriSpeech samples.
 - **Impact:** This limitation means the model may not perform as well with speech data from different accents or languages, reducing its applicability and usability in a global context.
 - **Real-Time Input Handling**
 - **Issue:** The model is not designed to process real-time input, which is a common requirement for ASR systems in real-world applications.

- **Impact:** This limitation restricts the model's use in dynamic, interactive applications where immediate feedback is crucial.
 - **Transfer of Errors**
 - **Issue:** If the ASR component of the model makes an error in transcribing speech, this error is passed on to the music recommendation model, leading to irrelevant recommendations.
 - **Impact:** This creates a cascading effect where a single error in speech recognition can significantly impact the quality of the music recommendations, potentially leading to poor user experience.
- **Model 2: Music Recommendation Component**
 - **Handling Small Errors in the Query**
 - **Issue:** The music recommendation model is not robust enough to handle small errors in the query, which can occur due to the imperfect nature of speech recognition.
 - **Impact:** This weakness means the model may not be able to provide accurate recommendations when the input query contains minor transcription errors, limiting its effectiveness in practical applications.

Possible Future Improvements

- **Expanding the Training Data:** Incorporating more diverse accents and languages into the training dataset can help the model generalize better to different speech patterns.
- **Real-Time Processing:** Developing a version of the model that can process real-time audio input could enhance its applicability in interactive applications.
- **Error Handling:** Implementing strategies to handle errors in the ASR output, such as re-processing the audio or using fallback mechanisms, can improve the robustness of the system.
- **Enhancing the Music Recommendation Model:** Improving the model's ability to handle minor errors in the query, possibly through more sophisticated matching algorithms or additional preprocessing steps, can improve the accuracy of recommendations.
- **Data Augmentation Techniques:**
 - Data augmentation is a technique that artificially increasing the size and diversity of the training dataset.
 - This is achieved through various transformations, such as time stretching, pitch shifting, adding noise, and textual modifications like synonym replacement and random insertion.

- By exposing the model to a wider range of data variations, data augmentation enhances the model's generalization capabilities, making it more robust and efficient in handling real-world data.

Code File Link : [Colab File](#)

References

<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8461690>

<https://www.simonwenkel.com/notes/ai/practical/audio/speech-to-text-GRUs-LSTMs.html#gru-gated-recurrent-unit-units>

https://d1wqtxts1xzle7.cloudfront.net/49255272/A_Review_on_Speech_Recognition_Technique20160930-21561-nv0vxq-libre.pdf?1475291980=&response-content-disposition=inline%3B+filename%3DA_Review_on_Speech_Recognition_Technique.pdf&Expires=1712943580&Signature=Fruo4n7AhpKj20qk6WpCfdsm~zJcrJ-LoQseijcrG2yu6PQdHvpr1HcVexaq8BVfvBrz~48bvHQ0DpJxs9SO-y2afliF2w1~3JmQDIBboltPHV1PhSCrlvA5r8oFFbpUZ2M5Dc1xwBk5WR0ynqT-FMU6SUADCnhzBtXT~FIWxuusuP-1i8P9H0iDgvh9wZ8~RXTNDspOeylKZ8Uqm4eW9HJMaElonsiurmev3EVX7vjgZJVaN0iPtVSL3EjUv-myc8ALHHdCKtnbVTIxM5qB2ZGSHYhHpx69FrO9mJJzB-MhTcldLKvFxfFRzY3~EZn7dkusJU1AltLrOIB0AOBw__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6857341>

<https://www.pinecone.io/learn/series/faiss/faiss-tutorial/>

<https://www.sbert.net/>

Contributions

Rishav Aich (B21AI029)

- Implemented model-2 (Music Recommendation model) (coding part)
- Prepared Report

Ashutosh (B21AI007)

- Implemented model-1 (coding part)
- Prepared Report

Saloni Garg (B21AI036)

- Literature review for model-1 architecture.
- Search and prepare the dataset for speech recognition.

Harsh Vardhan (B21BB012)

- Search and prepare the dataset for spotify million song dataset.
- Literature review for model-2 (music recommendation system)