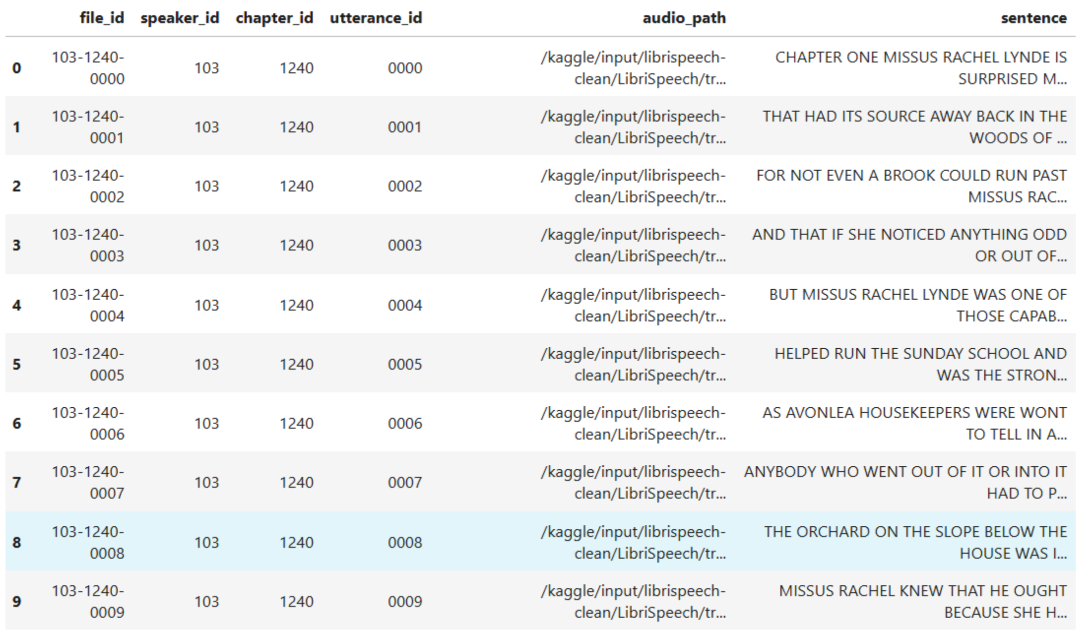
MID TERM REPORT

Anushka Tyagi
24112023

1) **Task Description**
   The objective of my project is to create a ***SUBTITLE GENERATOR*** from scratch. It involves taking a video file in the form of a Youtube video URL , extracting the audio, and generating subtitles that are synchronized with the video content. This involves audio processing, speech-to-text conversion, and synchronization of the generated text with the video.To achieve this, first the video is downloaded from Youtube URL using a tool like **Pytube** and then using a framework which is **FFmpeg** which extracts audio from video files.Then the audio is processed by generating a **spectrogram** which is a visual representation of frequencies of an audio signal as it varies with time. Spectrograms can be generated by applying **Fourier Transformation**. Processing audio into spectrograms converts it into a numerical representation, allowing machine learning models to interpret the data.The spectrogram data is processed using a hybrid **CNN-RNN** model.The model is trained with spectrogram inputs and character-sequence labels to generate subtitles.I am using **Keras** and **TensorFlow** to build and train the neural network model.The predicted subtitles will be then synchronized with the video and overlaid on it using **FFmpeg** to create subtitled video.
   Pytube does not allow downloading youtube videos which I discovered while using it so instead I will be using **YT-DLP** to fetch the video content from a YouTube URL.Also spectrogram is generated using the inbuilt features of **Librosa** library. These changes and alternatives were suggested to me by my mentors to optimize the performance of the subtitle generator.

2) **Dataset**
   The dataset I used for training my model is **LibriSpeech,**a large open-source speech dataset**.**Due to its large size I have chosen its subset **train-clean-100** so that the dataset can be easily processed. It consists of high-quality speech audio in **.flac** format organized hierarchically (speaker → chapter → utterances). Each audio file has a corresponding transcript stored in **.trans.txt**.
   I processed the LibriSpeech dataset in a simplified way to access its audio files and transcriptions, which are stored in a complex folder structure. The approach involved extracting file IDs to match audio files with their corresponding transcriptions. I then created a **Pandas DataFrame** containing columns for the file ID, speaker ID, chapter ID, utterance ID, audio file path, and the transcription.

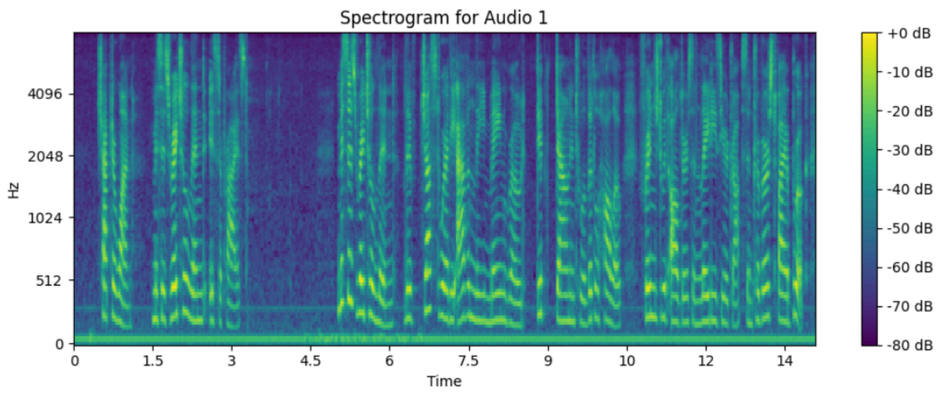
This method allows for efficient retrieval of both audio and text for further use addressing the dataset's complicated organization.



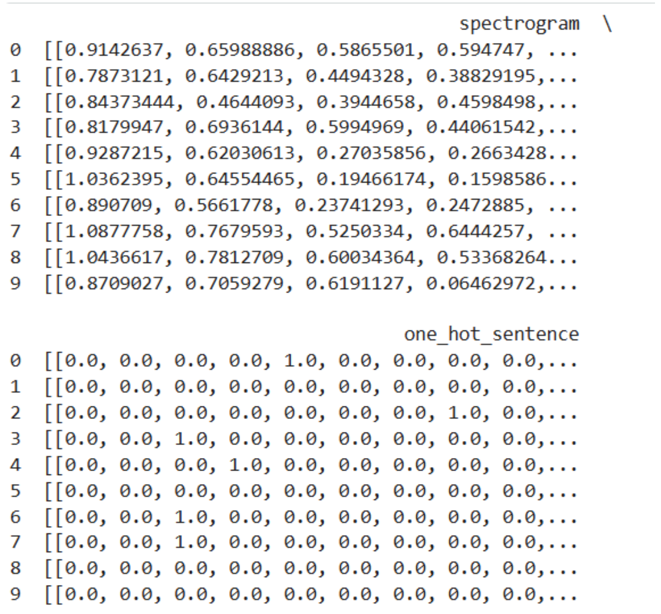
	file_id	speaker_id	chapter_id	utterance_id	audio_path	sentence
0	103-1240-0000	103	1240	0000	/kaggle/input/librispeech-clean/LibriSpeech/tr...	CHAPTER ONE MISSUS RACHEL LYNDE IS SURPRISED M...
1	103-1240-0001	103	1240	0001	/kaggle/input/librispeech-clean/LibriSpeech/tr...	THAT HAD ITS SOURCE AWAY BACK IN THE WOODS OF ...
2	103-1240-0002	103	1240	0002	/kaggle/input/librispeech-clean/LibriSpeech/tr...	FOR NOT EVEN A BROOK COULD RUN PAST MISSUS RAC...
3	103-1240-0003	103	1240	0003	/kaggle/input/librispeech-clean/LibriSpeech/tr...	AND THAT IF SHE NOTICED ANYTHING ODD OR OUT OF...
4	103-1240-0004	103	1240	0004	/kaggle/input/librispeech-clean/LibriSpeech/tr...	BUT MISSUS RACHEL LYNDE WAS ONE OF THOSE CAPAB...
5	103-1240-0005	103	1240	0005	/kaggle/input/librispeech-clean/LibriSpeech/tr...	HELPED RUN THE SUNDAY SCHOOL AND WAS THE STRON...
6	103-1240-0006	103	1240	0006	/kaggle/input/librispeech-clean/LibriSpeech/tr...	AS AVONLEA HOUSEKEEPERS WERE WONT TO TELL IN A...
7	103-1240-0007	103	1240	0007	/kaggle/input/librispeech-clean/LibriSpeech/tr...	ANYBODY WHO WENT OUT OF IT OR INTO IT HAD TO P...
8	103-1240-0008	103	1240	0008	/kaggle/input/librispeech-clean/LibriSpeech/tr...	THE ORCHARD ON THE SLOPE BELOW THE HOUSE WAS I...
9	103-1240-0009	103	1240	0009	/kaggle/input/librispeech-clean/LibriSpeech/tr...	MISSUS RACHEL KNEW THAT HE OUGHT BECAUSE SHE H...

3) Model Architecture

The subtitle generator model uses **CNN** and **RNN** to predict textual subtitles.The input for the neural network will be a **spectrogram**, which is a 2D array (matrix) of numbers representing the distribution of frequencies in an audio signal over time.Most of the information in human speech is contained within the 0–8 kHz range.So while generating spectrograms **maximum frequency** to be included is set to 8KHz to ensure focus only on the most relevant frequency range therefore reducing computational cost.Also **sampling rate** is the number of samples of audio captured per second, measured in Hertz (Hz).It is set to 22050 Hz for reduced computational load without significant loss of important audio information.



The training data inputs are the **spectrograms** of the audios which were present in the Librispeech dataset and the output for training the model are the corresponding **transcriptions** of the audios. They will be accessed from the pandas dataframe which was created for processing our dataset.Since our deep learning model can not understand the textual transcriptions, each character in the transcription (alphabets,spaces ,punctuation marks etc) will be converted into a **one-hot encoded** format by mapping of characters to indices.Here a dictionary is constructed where each unique character in the dataset is assigned a specific numeric index, resulting in a one-hot encoded representation of every sentence. The mapping of inputs and outputs is done using a dataFrame containing **normalized spectrograms** and their corresponding **one-hot encoded text representations**.


```
                                         spectrogram  \
0  [[0.9142637, 0.65988886, 0.5865501, 0.594747, ...
1  [[0.7873121, 0.6429213, 0.4494328, 0.38829195,...
2  [[0.84373444, 0.4644093, 0.3944658, 0.4598498,...
3  [[0.8179947, 0.6936144, 0.5994969, 0.44061542,...
4  [[0.9287215, 0.62030613, 0.27035856, 0.2663428...
5  [[1.0362395, 0.64554465, 0.19466174, 0.1598586...
6  [[0.890709, 0.5661778, 0.23741293, 0.2472885, ...
7  [[1.0877758, 0.7679593, 0.5250334, 0.6444257, ...
8  [[1.0436617, 0.7812709, 0.60034364, 0.53368264...
9  [[0.8709027, 0.7059279, 0.6191127, 0.06462972,...

                                    one_hot_sentence
0  [[0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0,...
1  [[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,...
2  [[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0,...
3  [[0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,...
4  [[0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0,...
5  [[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,...
6  [[0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,...
7  [[0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,...
8  [[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,...
9  [[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,...
```

Additionally, before training both inputs and outputs data is filtered to ensure spectrograms are 2D numpy arrays and sentences are non-empty lists/arrays.Also, both are padded as it ensures all spectrograms and encoded text have consistent shapes, enabling batch processing and alignment for model training.

The neural network begins with **convolutional layers (Conv2D**) to extract features from spectrograms. These layers are followed by **batch normalization** and **max-pooling** to reduce dimensionality and enhance feature learning. Afterward, the model flattens the data and applies dense layers to prepare it for **sequence modeling.** The model then uses a **Bidirectional LSTM laye**r to capture **temporal dependencies** in the data. The output layer, a **TimeDistributed Dense layer** with **softmax activation**, predicts character probabilities at each time step. The model is compiled with the **Adam optimizer** and **categorical cross-entropy loss function** and trained on the provided data. Finally, the model's performance is evaluated on a test set to check its accuracy and loss.

```
# Convolutional layers
x = Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)
x = BatchNormalization()(x)
x = MaxPooling2D(pool_size=(2, 2))(x)


x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D(pool_size=(2, 2))(x)

x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D(pool_size=(2, 2))(x)

# Flatten and reshape for RNN
x = Flatten()(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.3)(x)
x = Dense(output_sequence_length * num_characters, activation='relu')(x)
x = Reshape((output_sequence_length, num_characters))(x)
```

```
# LSTM layer for sequence modeling
x = Bidirectional(LSTM(256, return_sequences=True))(x)
x = Dropout(0.3)(x)

# Fully connected layer for character prediction
outputs = TimeDistributed(Dense(num_characters, activation='softmax'))(x)

# Compile the model
model = Model(inputs, outputs)
model.compile(
    optimizer=Adam(learning_rate=0.01),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

- ➔ *Sequence modeling is about working with data where the order matters, like a sentence or a time series. It involves predicting or understanding the next part of a sequence based on what came before it. For example, in speech-to-text, the model listens to a sequence of sounds and predicts the next word or letter.*
- ➔ *Temporal dependencies refer to the relationship between the current and past information in the sequence.Temporal dependencies in my data are the patterns of how each audio frame (spectrogram) and each word/character (in the text) are related to those that came before and after, and the model must learn these dependencies to make accurate predictions.*

The model predicts a sequence of one-hot vectors, where each vector corresponds to the predicted character at a particular time step.The predicted one-hot vectors are decoded into human-readable text using a function which maps each one-hot vector back to the corresponding character.

The true text labels (ground truth) are also decoded from the test set's one-hot encoded labels which were present in our dataframe to compare it with predictions made by our model. A metric like **Word Error Rate (WER)** is used to evaluate the performance of models by comparing the predicted text to the ground truth.A lower WER indicates better performance, meaning fewer errors in word predictions.

4) Implementation

Till now I have completed the following:

1. Studied the following concepts of Deep Learning from basics in detail:
    - What is Deep Learning & Where it is used.
    - What is a Perceptron & how to train it.
    - Loss Functions
    - Activation Functions
    - Gradient Descent
    - Multi Layer Perceptron
    - Optimizers
    - Convolutional Neural Networks
    - Ordinal Encoding & One Hot Encoding
    - Recurrent Neural Networks
    - LSTM (its architecture & the three types of gates)

2. Code for converting the Youtube video into audio form using the URL of the video.This is done using YT-DLP and FFmpeg.

```python
import os
import subprocess
from IPython.display import Audio, display
import librosa

def download_and_play_youtube_audio(url, output_path="output.mp3"):
    try:
        # Use yt-dlp to download the audio
        subprocess.run([
            'yt-dlp',
            '-x', '--audio-format', 'mp3',
            '-o', 'temp_audio.%(ext)s',
            url
        ], check=True)

        temp_file = "temp_audio.mp3"

        # Convert to the desired output path if needed
        if output_path != temp_file:
            os.rename(temp_file, output_path)
```

```python
        # Load the audio file using librosa
        audio_data, sr = librosa.load(output_path, sr=None)

        # Display audio player in Kaggle notebook
        print(f"Playing audio from: {output_path}")
        display(Audio(audio_data, rate=sr))

        return True

    except Exception as e:
        print(f"Error: {str(e)}")

url = "https://www.youtube.com/watch?v=IwrSFGfo9CE"
download_and_play_youtube_audio(url, "output.mp3")
```

3. Uploading the dataset and processing it in the form of a dataframe to easily access the audios along with its paths and corresponding transcriptions.

4. Generating spectrograms of the audios and encoding the transcriptions and creating a dataframe containing both.

```python
# Generate Mel spectrogram for an audio file
def generate_spectrogram(audio_path, target_sr=22050):
    audio, sr = librosa.load(audio_path, sr=None)
    audio_resampled = librosa.resample(audio, orig_sr=sr, target_sr=target_sr)
    spectrogram = librosa.feature.melspectrogram(y=audio_resampled, sr=target_sr, n_mels=128, fmax=
8000)
    spectrogram_db = librosa.power_to_db(spectrogram, ref=np.max)
    return spectrogram_db
```

```python
# One-hot encode text
def one_hot_encode_text(text, char_to_idx):
    one_hot = np.zeros((len(text), len(char_to_idx)), dtype=np.float32)
    for i, char in enumerate(text):
        if char in char_to_idx:
            one_hot[i, char_to_idx[char]] = 1.0
    return one_hot

# Create character-to-index mapping
unique_chars = set(''.join(df['sentence']))
char_to_idx = {char: idx for idx, char in enumerate(sorted(unique_chars))}
df['one_hot_sentence'] = df['sentence'].apply(lambda x: one_hot_encode_text(x, char_to_idx))
```

5. Filtering , Padding and Splitting the data into train,test and val.

   Currently I am working on training the model and also trying & researching on improving my neural network architecture for better predictions and a lower word error rate.

5) End-Term Goals

My End-Term Goals are to make a better and improvised version of my current model architecture for it to give perfect predictions and least word error rate.This may require certain changes in the current neural network architecture.The model also needs to be tested on a large data to see its performance.After this, I have to synchronize the generated subtitles with the video and overlay them on video using FFmpeg.Finally the SUBTITLE GENERATOR will be made by putting all the steps together which include: Converting the entered Youtube video URL into audio, generating the transcriptions(subtitles) for the audio which will be done by the model which was trained and tested on Librispeech dataset and finally synchronizing & overlaying the subtitles on the video.

6) References

- https://youtube.com/playlist?list=PLKnIA16_RmvYuZauWaPlRTC54KxSNLtNn&si=e2t_1lRMQ6aE16yk
- https://youtu.be/ZLIPkmmDJAc?si=wqpwAiOi7YF_51mP
- https://www.kaggle.com/code/heiswicked/libri-01
- https://github.com/nicknochnack/DeepAudioClassification/blob/main/AudioClassification.ipynb
- https://www.youtube.com/watch?v=dJYGatp4SvA
- https://towardsdatascience.com/audio-deep-learning-made-simple-part-1-state-of-the-art-techniques-da1d3dff2504

- https://github.com/vrnanshuman/speech-recognition-with-nn/blob/main/vui_notebook.ipynb
- https://www.sciencedirect.com/science/article/pii/S1877050924006227?ref=pdf_download&fr=RR-2&rr=8f9cd116fad98e86