Studying An ASR System and Making An Acoustic Speech Recognition Model

Introduction

The objective of this project was to learn how to create an ASR(automatic speech recognition) system and to create a deep learning-based acoustic model for speech recognition by employing Mel-frequency cepstral coefficients (MFCCs), convolutional neural networks (CNNs), recurrent neural networks (RNNs), and connectionist temporal classification (CTC) loss. We initially wanted to create something that would be helpful to those with disabilities, as a schizophrenic speech recognizer, but our team ran into a challenge with the level of expertise and grew concerns towards biasness and sensitivity on the topic itself. At first, we collected numerous videos that consisted of mainly interviews and clips of schizophrenic people entering a psychotic episode. We then thought that we went way over our heads with the idea and instead decided to study ASR, particularly Vosk's speech recognition API. We were inspired when I came across a video of a young man's project called LanguageLeapAI that creates your own Personal Multilingual AI Translator. The project is only limited to Japanese and German at the moment, but one of the python programs allows you to use your microphone with keyboard input and it then saves your voice in an audio file which is then sent to WhisperAI which runs ASR. After that the text is then translated with DeepL's REST API and sent to a Japanese Deep-Learning AI Voice Synthesizer to perform text-to-speech for daily use such as gaming or using other communications platforms. With little knowledge other than taking this course and CS-549 or Machine Learning with Professor Yang Xu here at San Diego State University, we definitely ran into several roadblocks during this project and couldn't fully develop our ideas. In the end we were only able to create an acoustic model that defines a deep learning model for speech recognition using the LibriSpeech dataset, and also created a simple translator using Vosk's language models. Speech recognition is a crucial task in natural language processing with various applications, such as voice assistants, transcription services, and speech-to-text conversion. Recent advancements in deep learning have facilitated the development of end-to-end models capable of learning features directly from raw audio signals.

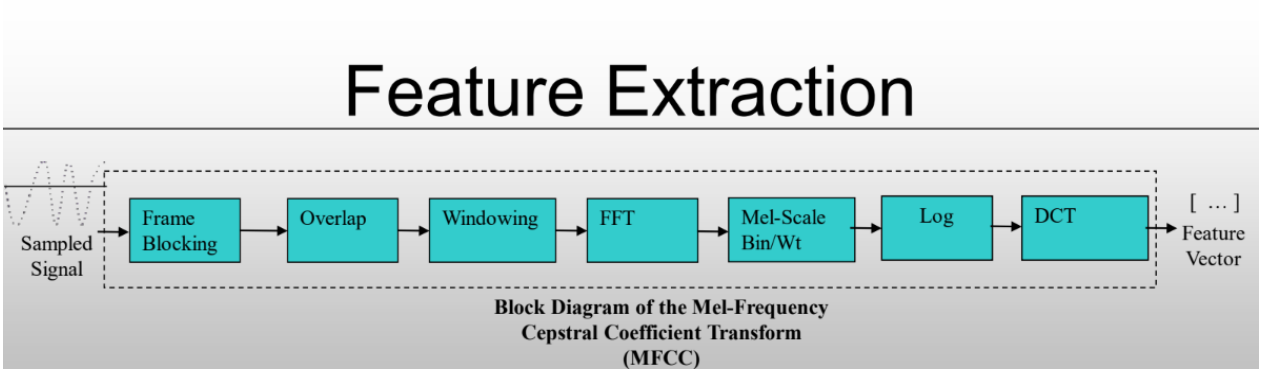
Problem

Our team's focus for this project is to look at the development challenges in designing and implementing an ASR system. We wanted to develop a system that can accurately recognize and transcribe speech from one language in real-time, and then translate the transcribed text into another language, also in real-time. The ultimate goal is to create a translator that enables communication across different languages and cultures around the world. The primary problem is understanding how we can utilize deep learning techniques, focus on acoustic modeling, and figure out how to accurately convert spoken language into text while handling these challenges associated with diverse phoneme classification and variable-length speech signals.

Background

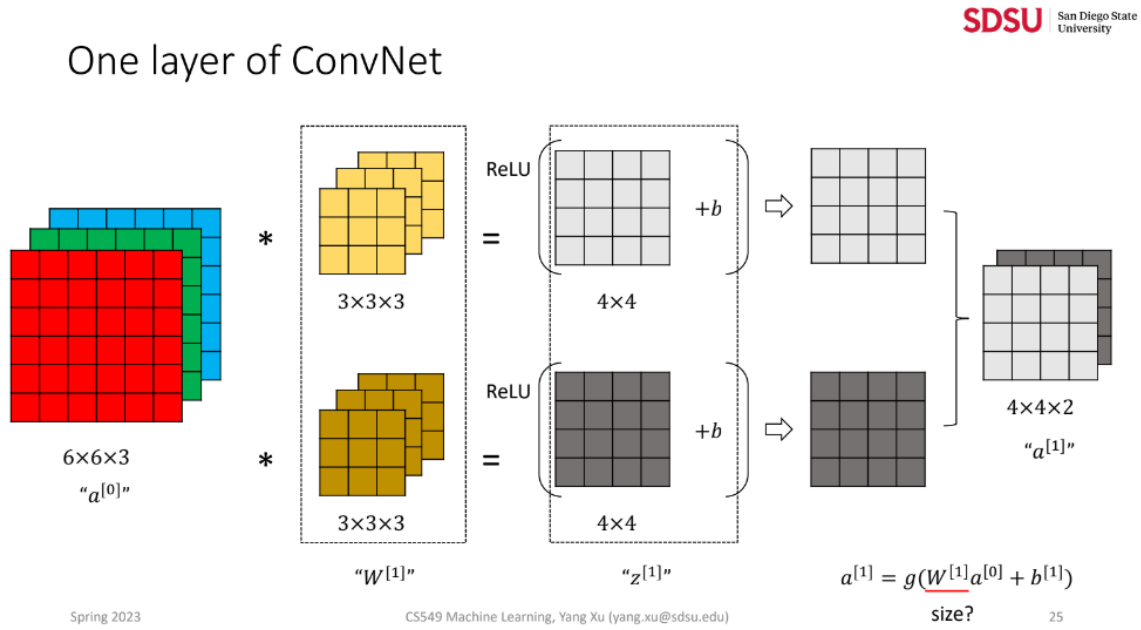
Speech recognition entails converting spoken words into written text, typically involving several stages: feature extraction, acoustic modeling, language modeling, and decoding. Feature extraction transforms the raw audio signal into a set of features that encapsulate the speech signal's spectral characteristics. One prevalent feature extraction technique is Mel-frequency cepstral coefficients (MFCCs) which is rooted in the perceptual properties of human hearing. The MFCC is the inverse of the log of the short-term power spectrum of the original signal(Konopka,

Lecture 3, 22).



Source: Courtney Konopka

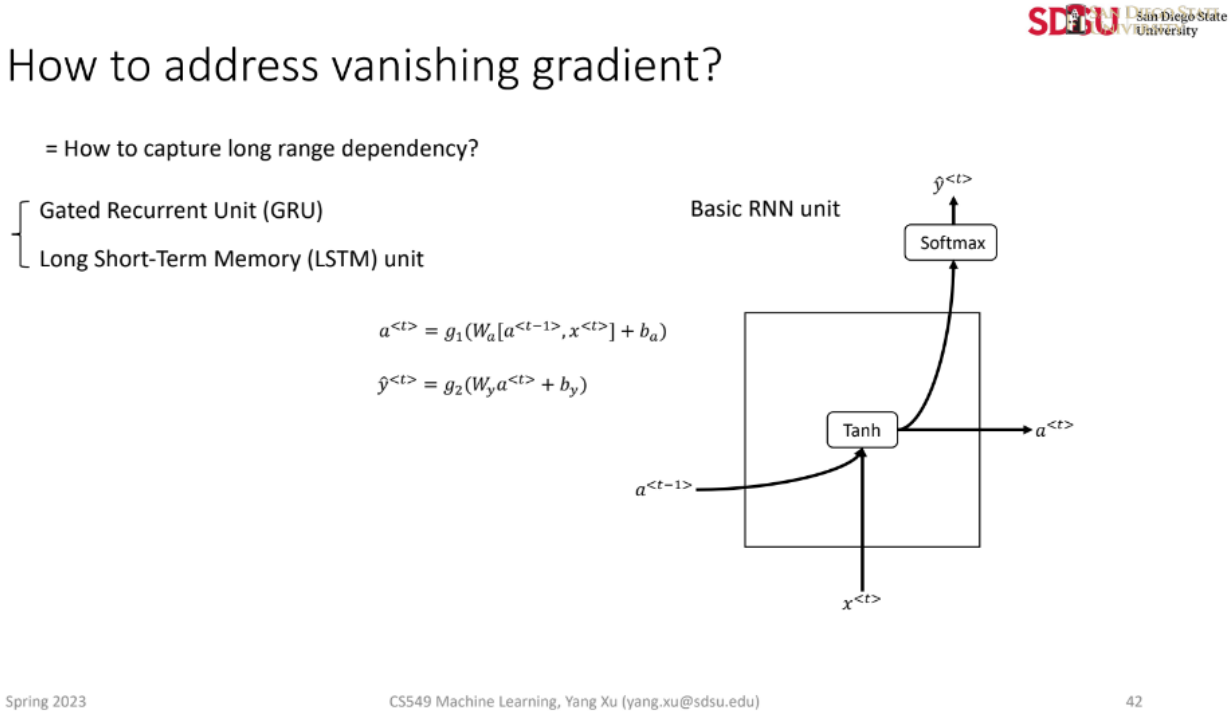
Convolutional neural networks (ConvNet/CNN) are a type of deep learning model adept at processing images and other spatial data as they are more commonly used in vision tasks(Xu, Optimization of Training Deep Neural Networks Lecture, 25). In speech recognition, CNNs can learn high-level representations of spectral features. The formula for a single convolutional layer can be expressed as:



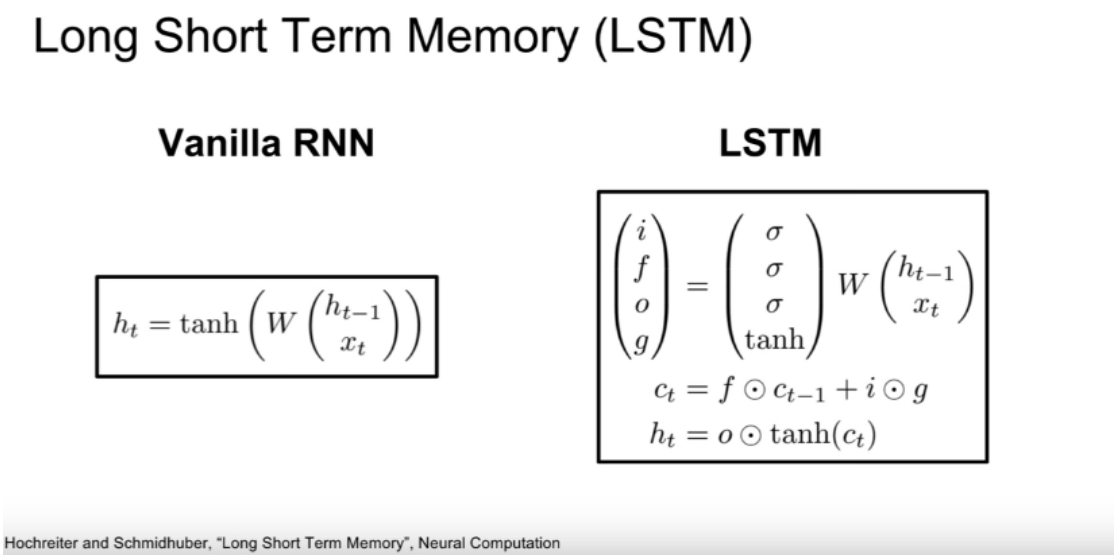
Source: Yang Xu

In which in our case for our model architecture:
'W' is a set of learnable filters(weights) with dimensions (filter_height, filter_width, input_channels, output_channels)
'b' is a set of biases with dimensions (output_channels,)
'*' represents the convolution operation
'a' is the pre-activation and activation outputs

Recurrent neural networks (RNNs) are another deep learning model type, ideal for processing sequential data and are typically applied in language modeling(Xu, Recurrent Neural Network Lecture, 12). End-to-end training techniques like Connectionist Temporal Classification(CTC) enable the training of RNN for tasks where the alignment between input and output is unknown(Graves, Mohamed, Hinton, 1). However, the vanilla RNN alone is not good at capturing long-term dependencies, in which they have vanishing gradient issues(Xu, Recurrent Neural Network Lecture, 39). The combination of Long Short-Term Memory(LSTM) has shown to address the vanishing gradient because it uses purpose-built memory cells to store information, which is better at finding and exploiting long range context(Graves, Mohamed, Hinton, 1).



Source: Yang Xu



Source: Hochreiter and Schmidhuber

Connectionist temporal classification (CTC) is a loss function commonly used in speech recognition for training, in which it is usually combined with CNN or RNN to build ASRs to handle the variability and ambiguity of speech data(Hannun, 2017). CTC enables the model to output variable-length sequences of symbols, which is essential for addressing the variable-length nature of speech signals(Jurafsky & Martin, 10). The formula to train a CTC-based ASR system as follows:

$$L_{CTC} = \sum_{(X,Y)\in D} -\log P_{\text{CTC}}(Y|X)$$

Source: Jurafsky & Martin

In which you use a special CTC loss function along with negative log-likelihood loss. As a result, the loss for the complete dataset D can be computed by adding up the negative log-likelihoods of the accurate output Y for each input X(Jurafsky & Martin, 10). In our code we used:

$$L_{CTC} = \quad -\sum_{y\epsilon Y} p(Y|X)$$

where Y is the set of all possible transcription sequences, X is the input feature sequence, and p(Y|X) is the probability of the transcription sequence Y given the input feature sequence X (Hannun, 2017). The CTC loss function calculates the negative log probability of the correct transcription sequence given the input features. It is used to adjust the weights and biases of the network to minimize the loss.

Dataset

We decided to use the LibriSpeech dataset for our project because it is an open-source corpus based on public domain audio books that is carefully segmented and aligned. It consists of

many hours of audio recordings and their corresponding transcriptions. The dataset is divided into subsets based on audio quality and background noise levels. For this project, we used the "train-clean-100" subset, comprising 100 hours of 16kHz read English speech due to its smaller size for training. For each audio file, it extracts Mel-spectrogram features using the librosa library, which are then used as inputs to the deep learning model. Alongside this, it reads the corresponding transcriptions of the audio files. The labels for the data are the speaker IDs. The extracted features are padded or truncated to a fixed length to ensure consistent input size for the model. The labels are converted to one-hot encoding for training the neural network. We preprocessed the audio recordings by extracting MFCCs with a 25 ms window size, a 10 ms hop size, and 128 mel frequency bins because they are common choices for MFCC extractions. It allows a balance between capturing enough temporal information in the signal while still maintaining a reasonable level of computational complexity(Piconepress, 3.1.2 Overview: Frame-Based Processing). 128 mel frequency bins is sufficient to capture the important frequency content of the speech signal while avoiding the introduction of too much unnecessary information into the feature representation. The MFCCs were normalized to have zero mean and unit variance to ensure that the input features have similar scales and ranges. The dataset was divided into training and validation sets at an 80:20 ratio which should provide enough data for training while still reserving enough data for model evaluation.

Methodology

The sequence modeling component models the temporal dynamics of the speech signal and predicts the sequence of linguistic units. The architecture of the acoustic model in our code consists of three main components: feature extraction, sequence modeling, and decoding. It was selected based on the unique strengths of each layer type and their suitability for speech recognition tasks. The project's model architecture consists of a series of convolutional and recurrent layers, followed by a CTC loss layer. This combination of layers was chosen to capture both the spatial and temporal information in the input data, which is what we wanted for accurate speech recognition. The input to the model is a sequence of MFCC frames, with the output being a sequence of phonemes. We implemented the model using TensorFlow and Keras.

The model architecture includes the following layers:

1. Conv2D layer with 32 filters, a (3, 3) kernel size, and ReLU activation
2. BatchNormalization layer
3. Conv2D layer with 64 filters, a (3, 3) kernel size, and ReLU activation
4. BatchNormalization layer
5. Dropout layer with a 0.5 rate
6. TimeDistributed layer with a Flatten layer
7. Bidirectional LSTM layer with 256 units and return sequences
8. Dropout layer with a 0.5 rate
9. Dense layer with a softmax activation

Our team studied and referenced homework assignments and lectures from CS-549 Machine Learning with Professor Yang Xu and other outside sources as cited to construct our model. For deep neural networks, gradients can get extremely big or small. We chose Conv2D layers with ReLU activation because Convolutional layers (Conv2D) are effective at capturing

local patterns and spatial hierarchies within the input data. It is used to extract the local features from the input MFCC spectrograms. The use of Conv2D layers is appropriate in this case because of the 2D nature of the MFCC spectrograms, which can be thought of as 2D images(Nandi, 2021). The filters used in the Conv2D layers learn to recognize various patterns in the spectrograms.

The ReLU activation function helps learn high-level representations of spectral features in the MFCC frames. It sets all negative input values to zero, and for positive values, it returns the input value directly. This helps to prevent the saturation of neurons that can occur with other activation functions. The use of ReLU can also improve the training speed of the model by reducing the likelihood of gradient vanishing and exploding, which can slow down the convergence of the model during training(Xu, Optimization of Training Deep Neural Networks Lecture, 51). BatchNormalization layers were used to normalize the input data within each mini-batch. It makes the training more efficient as we are trying to accelerate the training process and improve model performance by reducing the risk of vanishing or exploding gradients(Loffe & Szegedy, 2015).

Dropout layers were necessary as they are a regularization technique that helps prevent overfitting by randomly dropping out neurons during training(Xu, Assignment 8: Optimization of Deep Neural Networks). In his assignment, he taught us that the larger the droprate used, the lower test accuracy will be received, and training converges faster with Adam. This forces the model to learn more robust representations and improves its generalization capabilities. The Time Distributed layer with a Flatten layer combination allows the model to apply the Flatten operation independently to each time step in the input sequence as stated by the Keras API guide. It helps in connecting the output of the convolutional layers to the recurrent layers. The Bidirectional LSTM layer processes the input data in both forward and backward directions, enabling the model to capture context from both past and future time steps(Graves, Mohamed, Hinton, 2). This is very useful in speech recognition tasks where context from surrounding phonemes can impact recognition accuracy. The dense layer with softmax activation serves as the output layer of the model, producing a probability distribution over the phoneme classes. The CTC loss function is used during training to optimize the model parameters, and the softmax function is applied during inference to produce the final output probabilities for each time step(Jurafsky & Martin, 12). The CTC decoding algorithm is then applied to map the output probabilities to the final output sequence, accounting for the possible alignment between the input and output sequences(Graves, Mohamed, Hinton, 2). Softmax activation ensures that the output probabilities sum up to one, enabling the model to make a prediction for each time step in the input sequence.

The architecture combines these layers to create a model that can learn spectral and temporal patterns from the MFCC frames, resulting in improved speech recognition performance. The architecture is supposed to be adaptable and can be modified or extended in future work to include other advanced deep learning techniques, such as attention mechanisms.

Results

Source Codes:

Training Acoustic Model Code
About:

The proposed algorithm is based on Natural Speech Processing methods. This code is utilizing the machine learning techniques we learned with Yang Xu for Machine Learning- methods that we chose specifically to optimize the training for our model in .h5 format to be read in by our application. We chose to use Convolutional Neural Networks (CNNs) for the model definition, and LSTM (Long Short Term Memory) for recognizing spoken words in our audio files. To making our training more efficient, we trained the model with Adam which is an adaptive optimization and stochastic gradient descent method that is built into PyTorch(tensorflow.org, API Docs: Adam). Adam will help us set our model's trainable variables and parameters and minimize loss to our model. CTC (Connectionist Temporal Classification) is also used in hand to monitor the process and performance of the training and validation sets.CTC is an algorithm used to train sequence problems like speech and handwriting recognition. It will help with evaluating the quality of our model.

Most importantly, the model definition is the heart of the code with the model being created with Keras, an open-source deep learning framework written in Python that is designed to enable quick experimentation with deep neural networks and it will help us make model grouping layers into an object with training/inference features(keras.io, API Models Documentation). The model architecture consists of the following layers: Conv2D, BatchNormalization, Dropout, TimeDistributed, Bidirectional LSTM, and Dense layers explained in detail of their meanings previously.

Then, our training in particular will take a look and output the extracted features, labels, and transcriptions. Features are the inputs that will go into the model and are the aspects of the audio captures using MFCCs such as the frequency, intensity and characteristics that can be used to identify the patterns and differences between the audio samples. The labels are the outputs of the model and are what the model will try to predict or classify. These are our speaker IDs and will be held in y. The transcriptions are the textual files of the content in the Librispeech data and are stored alongside the features and labels. They will aid in our ability to transcribe the spoken speech into text in our translation app.
This code will also record, plot training and validation accuracy and loss over epochs. It is advised by Professor Xu that plotting it will allow us to evaluate the model over the training and validation data and analyze the accuracy performance.
Designed for reusability, this code can be used to train much larger datasets as we intend to do in the future.

Pseudocode:

1. Import our necessary Libraries
2. Set the Data Directory
3. Feature extraction function,
    a. Load the audio files
    b. Compute the Mel-Spectrogram
    c. Convert Mel-spectrogram to decibel units
    d. Return the converted Mel-spectrogram
4. Data loading function
    a. Initialize empty list for our extracted features
        i. Initialize for features, labels, and transcriptions

ii. For each audio file read from directory
    1. Extract speaker ID
    2. Read speaker transcription
    3. If length of feature is less than the max_len
        a. Pad features with zeros
    4. If length of features is more than max_len
        a. Truncate features
    5. Append features to X, labels to y, and transcription to transcriptions to store them
    6. Return Features
5. Load data function call
6. Split data into training and validation sets
7. Model creation function
    a. Initialize model with input shape
    b. Add Conv2D, BatchNormalization, Dropout, TimeDistributed, Bidirectional LSTM, and Dense layers
    c. Apply softmax activation function to Dense layer outputs
    d. CTC decoding to the softmax outputs
    e. Return model
8. Create model
9. Compile model with Adam optimizer and CTC loss function
10. Train model on training data and validate on validation data
11. History plotting function
    a. Plot training and validation accuracy over epochs
    b. Plot training and validation loss over epochs
12. Evaluate model over the training and validation data
13. Output training and validation accuracy
14. Plot training history
15. Save the trained model to directory

```
!pip install tensorflow
!pip install librosa
import os
import numpy as np
import librosa
import tensorflow as tf
from tensorflow.keras.layers import Dense, Dropout, Conv2D, Flatten, BatchNormalization,
Input
from tensorflow.keras.models import Model
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
from tensorflow.keras.layers import TimeDistributed, LSTM, Bidirectional, Activation,
Lambda
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.backend import ctc_batch_cost
```

```
import re

# Directory where the librispeech data is downloaded
data_dir = 'C:\\Users\\jenny\\Notebooks\\LibriSpeech\\train-clean-100'
train_path = os.path.join(data_dir, 'train-clean-100')

def extract_features(file_path, sample_rate=16000, n_mels=128):
    audio, _ = librosa.load(file_path, sr=sample_rate, res_type='kaiser_fast', dtype=np.float32)
    print(type(audio))  # Debugging: print the type of audio

#Apply the discrete cosine transform to obtain the final coefficients
    mfccs = librosa.feature.melspectrogram(y=audio, sr=sample_rate, n_mels=n_mels)
    mfccs = librosa.power_to_db(mfccs).T
    return mfccs

def load_data(path, sample_rate=16000, n_mels=128, max_len=300):
    X, y, transcriptions = load_data(train_path)

    for root, dirs, files in os.walk(path):
        print(f"Processing directory: {root}")
        for file in files: # For each audio file
            if file.endswith('.flac'):
                file_path = os.path.join(root, file)
                try:
                    label = int(file.split('-')[0])
                    transcription_file = os.path.join(root, file.replace('.flac', '.txt'))
                    with open(transcription_file, 'r') as f:
                        transcription = f.read().strip()
                    features = extract_features(file_path, sample_rate, n_mels)
                    if features.shape[0] < max_len:
                        padding = np.zeros((max_len - features.shape[0], n_mels))
                        features = np.vstack((features, padding))
                    else:
                        features = features[:max_len, :]
                    X.append(features)
                    y.append(label)
                    transcriptions.append(transcription)
                except Exception as e:
                    print(f"Error processing {file_path}: {e}")
                    transcriptions.append(None)
    X, y = np.array(X), np.array(y)
    transcriptions = np.array(transcriptions)
    X = X[..., np.newaxis]
    y = to_categorical(y)
    return X, y, transcriptions
```

```
X, y, transcriptions = load_data(train_path)

print("Shape of X:", X.shape)
print("Shape of y:", y.shape)
print("Shape of transcriptions:", transcriptions.shape)

# Splitting our data for training and validation
y = to_categorical(y)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
X_train = X_train[..., np.newaxis]
X_val = X_val[..., np.newaxis]

# Defining the model layers with keras
def create_model(input_shape, num_classes):
    inputs = Input(shape=input_shape)
    x = Conv2D(32, kernel_size=(3, 3), activation='relu')(inputs)
    x = BatchNormalization()(x)
    x = Conv2D(64, kernel_size=(3, 3), activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)
    x = TimeDistributed(Flatten())(x)
    x = Bidirectional(LSTM(256, return_sequences=True))(x)
    x = Dropout(0.5)(x)
    x = Dense(num_classes)(x)
    outputs = Activation('softmax')(x)
    outputs = Lambda(lambda x: tf.keras.backend.ctc_decode(x,
input_length=tf.ones(tf.shape(x)[0])*tf.shape(x)[1], greedy=True)[0][0])(outputs)

    return Model(inputs=inputs, outputs=outputs)

input_shape = X_train.shape[1:]
num_classes = y_train.shape[1]
model = create_model(input_shape, num_classes)
model.compile(optimizer=Adam(), loss=ctc_batch_cost)
model.summary()

epochs = 25
batch_size = 32
history = model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs,
validation_data=(X_val, y_val))

def plot_history(history):
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='train')
```

```
    plt.plot(history.history['val_accuracy'], label='validation')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='train')
    plt.plot(history.history['val_loss'], label='validation')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.show()

# Evaluate the model
_, train_acc = model.evaluate(X_train, y_train, verbose=0)
_, val_acc = model.evaluate(X_val, y_val, verbose=0)
print(f'Train accuracy: {train_acc * 100:.2f}%, Validation accuracy: {val_acc * 100:.2f}%')

# Plot the training history
plot_history(history)


# Save the trained model to path
model_save_path = os.path.join(os.getcwd(), 'trained_model1.h5')
model.save(model_save_path)
print(f"Model saved to: {model_save_path}")
```

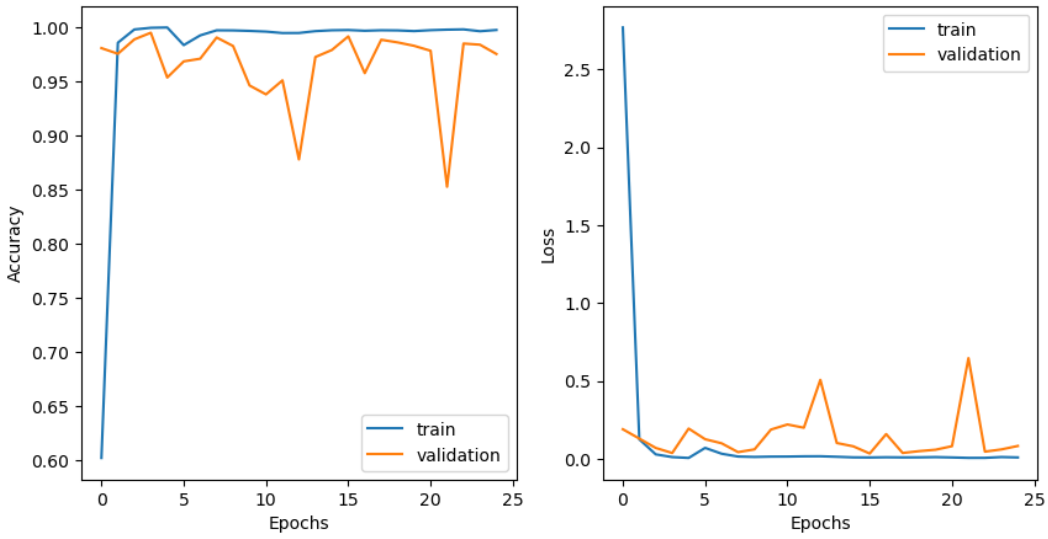
The Training Progress

```
                transcription = tf.keras.backend.get_value(transcription)[0]

                # Print the transcription for the segment
                print(f'Transcription for {audio_file} (segment): {transcription}')

%matplotlib inline
```

```
Transcription for C:\Users\jenny\Notebooks\LibriSpeech\train-clean-100\19\198\testing\19-198-0007.flac (segment): [8.4539
984e-09 7.2713484e-09 8.9128331e-09 ... 9.3790407e-09 7.7222317e-09
 2.3085269e-05]
1/1 [==============================] - 0s 116ms/step
Transcription for C:\Users\jenny\Notebooks\LibriSpeech\train-clean-100\19\198\testing\19-198-0007.flac (segment): [8.4256
007e-09 7.2467721e-09 8.8828100e-09 ... 9.3469117e-09 7.6966886e-09
 2.3206474e-05]
1/1 [==============================] - 0s 129ms/step
Transcription for C:\Users\jenny\Notebooks\LibriSpeech\train-clean-100\19\198\testing\19-198-0007.flac (segment): [8.4324
254e-09 7.2526833e-09 8.8897503e-09 ... 9.3545358e-09 7.7024822e-09
 2.3054685e-05]
1/1 [==============================] - 0s 122ms/step
Transcription for C:\Users\jenny\Notebooks\LibriSpeech\train-clean-100\19\198\testing\19-198-0007.flac (segment): [8.4668
335e-09 7.2827087e-09 8.9270458e-09 ... 9.3939070e-09 7.7342213e-09
 2.3035391e-05]
1/1 [==============================] - 0s 121ms/step
Transcription for C:\Users\jenny\Notebooks\LibriSpeech\train-clean-100\19\198\testing\19-198-0007.flac (segment): [8.4495
486e-09 7.2674795e-09 8.9078362e-09 ... 9.3738350e-09 7.7175626e-09
 2.3032557e-05]
```

In [19]:

```
import os
import numpy as np
import librosa
import tensorflow as tf
from tensorflow.keras.models import load_model
from tensorflow.keras.utils import to_categorical

# Load the trained model
model = load_model('trained_model.h5')

# Define the LibriSpeech dataset directory
```



Output Graph from Librosa's train-clean-100 data:
From the displayed training and validation accuracy and loss over the epochs, we analyzed from the validation lines struggles to stabilize in the Accuracy Graph compared to the training data. The Training accuracy keeps increasing and maintaining a high accuracy, but the validated accuracy increases and decreases often over the epochs. This tells us that the model is overfitting still despite our efforts to implement L2 regularization and dropout when it comes to

accuracy. The model we trained with train-clean-100 data is able to learn the data well but it has lower and unstable accuracy on the validation data meaning the model won't do that well on data it hasn't seen yet(Xu, Chapter 12: Convolutional Neural Network Lecture, 47). The same can be said for the Loss output as the validation loss spikes occasionally while the loss is low and constant for the training function.
Improving the model:
To address these issues, we could try to improve it with techniques such as L1, L2 regularization or dropout to prevent severe overfitting with the cost of the loss function for larger weights(Xu, Chapter 11: Optimization of Training Deep
Neural Networks Lecture, 25). These regularization techniques add a term to the loss function that is specific to the weight vector and encourages the model to have constraints to its network's complexity and improve its generalization of data. Our dropout tried to maintain stability but fell through at progressively higher epochs because it could not generalize the data well enough with the small set of data. If regularization does not work, we also must consider taking considerably more time to train a much larger dataset so it can help the model learn better and generalize well. This can be done as well with data augmentation to artificially increase the size of the training set to help improve the model's performance. However, we have yet to learn data augmentation techniques and were not taught in detail how to implement it in our Machine Learning Course. Ideally, we would like the model to be output as a .conf or .mdl file to match Vosk's trained models' format. Our code would need to have a trained language model alongside the acoustic model together to do so.

Predicting and Visualizing Audio with our Model Code

About:
A script that loads our pre-trained machine learning model using TensorFlow and Keras for the visualization waveform. It processes an audio file from the LibriSpeech dataset by extracting its Mel-Spectrogram features with Librosa and makes a prediction with the model based on these features. It visualizes the waveform and Mel spectrogram of the audio file using Matplotlib. This prediction output can be used to represent insight into the contents of the audio file including the speaker's biometric identity, sound classification and speech recognition. For our case, we want to use it for a speech recognition model to predict a transcription of speech in the audio file so it can be used to convert speech to text for our application. Assuming that these values are in scientific notation, these tiny numbers are very small probabilities. Jasmine and I realize that these probabilities' positions in the array could corressspond to a different varying word or phoneme given our file.

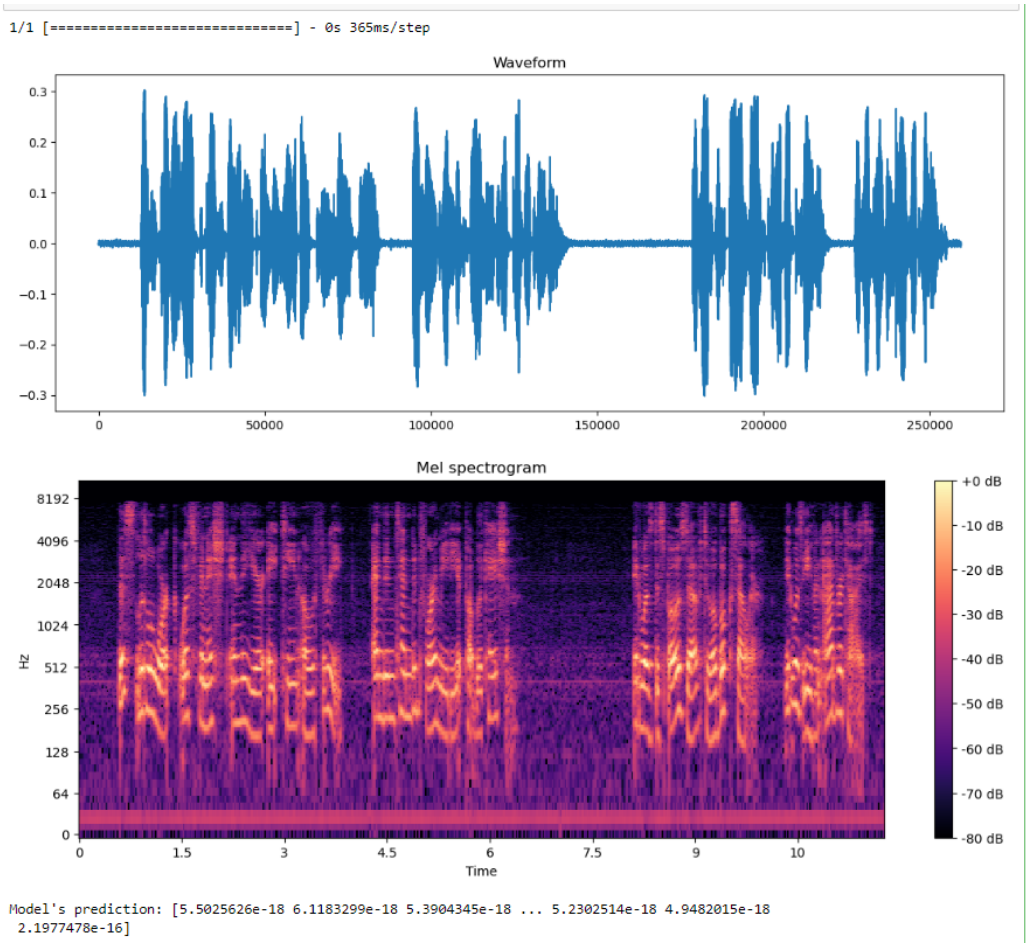
Pseudocode:

1. Import our necessary libraries
2. Feature extraction function(Exactly the same as the one in our Model)
    a. Load the audio files
    b. Compute the Mel-Spectrogram
    c. Convert Mel-spectrogram to decibel units
    d. If length of feature is less than the max_len
        i. Pad features with zeros

        e. Else if length of features is more than max_len
            i. Truncate features
        f. Return the converted Mel-spectrogram
    3. Load trained model
    4. Load audio file from library
    5. Extract Mel-Spectrogram features from audio file
    6. Use trained model to make prediction of extracted features
    7. Visualize audio file's waveform with matplotlib function
    8. Visualize audio file's Mel-Spectrogram with librosa's specshow function
    9. Print model prediction(MUST BE MODIFIED FOR SPECIFIC CONTEXT)

```
from tensorflow.keras.models import load_model
import numpy as np
import librosa
import matplotlib.pyplot as plt

# Function to extract features from a random audio file we pulled from the libraspeech data set
def extract_features(file_path, sample_rate=16000, n_mels=128, max_len=300):
    audio, _ = librosa.load(file_path, sr=sample_rate, res_type='kaiser_fast', dtype=np.float32)
    # Extracting the Mel-Spectrogram features
    mfccs = librosa.feature.melspectrogram(y=audio, sr=sample_rate, n_mels=n_mels)
    # Convert spectrogram to decibels
    mfccs = librosa.power_to_db(mfccs).T

    # If the shape of the MFCCs don't meet max length requirements add padding
    if mfccs.shape[0] < max_len:
        padding = np.zeros((max_len - mfccs.shape[0], n_mels))
        mfccs = np.vstack((mfccs, padding))
    else:
        # Else truncate our MFCCs
        mfccs = mfccs[:max_len, :]
    return mfccs[np.newaxis, ..., np.newaxis]

# Load the trained model from our notebooks directory
model = load_model('trained_model.h5')

# Load our random audio file to sample
audio_file =
'C:\\Users\\jenny\\Notebooks\\LibriSpeech\\train-clean-100\\19\\198\\19-198-0001.flac'
y, sr = librosa.load(audio_file)

# Extract the features from the .flac audio file
features = extract_features(audio_file)

# Make a prediction using the model
predictions = model.predict(features)
```

```
# Visualize the audio file's waveform using matplotlib's plot function
plt.figure(figsize=(14, 5))
plt.plot(y)
plt.title('Waveform')
plt.show()

# Visualize the audio file's mel spectrogram
D = librosa.amplitude_to_db(np.abs(librosa.stft(y)), ref=np.max)
# Define the plot size to a reasonable one
plt.figure(figsize=(14, 5))
librosa.display.specshow(D, sr=sr, x_axis='time', y_axis='log')
plt.colorbar(format='%+2.0f dB')
plt.title('Mel spectrogram')
plt.show()

# Print out the model's prediction
print("Model's prediction:", predictions[0])
```



In this output, we want to look at how the model's prediction, waveform, and Mel spectrogram visualizations can provide us insight into the performance and processing of a speech

recognition model. Our output of the model's prediction: [5.5025626e-18 6.1183299e-18 5.3904345e-18 ... 5.2302514e-18 4.9482015e-18 2.1977478e-16], is a prediction array from our trained.h5 neural network model and at first we were lost on how to interpret these values and assumed that

Real Time Translation with ASR Application Code

About:
This code takes inspiration from github user, skabbit's Vosk Speech Recognition Toolkit as we referenced how he initialized his microphone/input device to his application and import Vosk models from PC to his coding environment to setup our language recognizer with our models(skabbit, github: Vosk-API). This setup helped us tremendously in configuring the speech recognition bindings for Python to be readily used. The scaling of Vosk models makes them usable for small applications such as ours which is intended for the user to recognize audio input from their microphone using the import PyAudio library and it will translate their voice into their desired language.
The program is written so that it recognizes the language selected, outputs the transcription and the translated transcription from the desired language relying on Python's translator to map the models to the Vosk speech recognition models for searching. It will also generate an audio file of the translated transcription using the Text-To-Speech API for playback after successful translation. The button widgets are from python's own internal widget library ipywidget and Python and we referenced their readily available documentation on how to make our buttons work. The code needs further error handling when the user would like to clear their transcriptions or re-record their audio.

Pseudocode:

1. Initialize PyAudio and Vosk models:
    a. Import modules for
    b. Initialize PyAudio
    c. Define path of our language models
2. Setup recognizer function for given language
    a. Take language, loads corresponding model
    b. Initialize Kaldi recognizer
3. Create buttons
    a. Button for Record
    b. Button for Stop
4. Audio stream callback function
    a. Take raw audio data input
    b. Convert to expected format
    c. Feed it to recognizer
5. Event handlers function for buttons
    a. Functions for Record
        i. On click() start recording audio stream
        ii. Send audio stream to callback
    b. Functions for Stop

i. On click() stop audio stream
ii. Read audio stream and retrieve recognize text
iii. Translate to target language with translation function

c. Define translation function
   i. Translate given text to a target language using the translate module.
d. Set up GUI elements
   i. Define dropdown for language selection
   ii. Define output box for displaying recognized text
   iii. Define button for playing the translation,
   iv. Define an output box for audio.
e. Event handler for the "Play Translation" button
   i. Function to handle the "Play Translation" button click
   ii. Will take translated text and convert to speech using gTTS
   iii. Play the speech
f. Set up Text-to-speech engine
   i. Initialize a pyttsx3 engine
   ii. Set up a dropdown for voice selection.
g. Display all widgets

```
!pip install gtts
!pip install scipy
!pip install translate
!pip install ipywidgets
!pip install vosk
!pip install pyaudio
!pip install pandas
!pip install numpy
```

```
# Install pyaudio from http://people.csail.mit.edu/hubert/pyaudio/
# Find audio device index using this code
import pyaudio
p = pyaudio.PyAudio()
for i in range(p.get_device_count()):
    print(p.get_device_info_by_index(i))

p.terminate()

import pyaudio
p = pyaudio.PyAudio()
print(p.get_device_count())
for i in range(p.get_device_count()):
    print(p.get_device_info_by_index(i))
```

```
import pyaudio
p = pyaudio.PyAudio()
info = p.get_host_api_info_by_index(0)
```

```
numdevices = info.get('deviceCount')
for i in range(0, numdevices):
        if (p.get_device_info_by_host_api_device_index(0, i).get('maxOutputChannels')) > 0:
            print("Output Device id ", i, " - ", p.get_device_info_by_host_api_device_index(0,
i).get('name'))
#improved way to get microphone input when retrying
```

```
!pip install google.cloud
!pip install pyttsx3
!pip install gtts
!pip install translate
```

```
###import pyaudio
###def callback(in_data, frame_count, time_info, status):
###    print("Received audio data")
###    return (in_data, pyaudio.paContinue)
###p = pyaudio.PyAudio()
###stream = p.open(format=pyaudio.paInt16, channels=1, rate=16000, input=True,
frames_per_buffer=16000, stream_callback=callback)
###stream.start_stream()
## no longer needed for retrieving old audio data
```

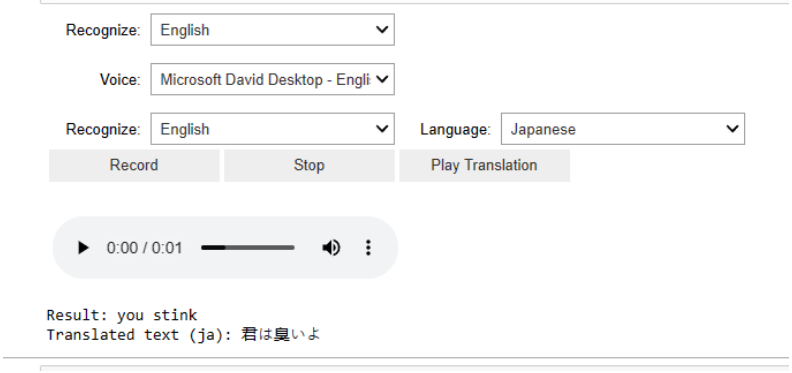
```
# Our necessary imports
import pyaudio
import threading
import vosk
import ipywidgets as widgets
import json
import os
import subprocess
import time
import scipy.signal
from translate import Translator
from gtts import gTTS
from IPython.display import Audio
import io
import pyttsx3

# initialize the vosk models with the file directories that hold them
vosk_model_names = {
    "en": "C:\\Users\\jenny\\Notebooks\\vosk-model-en-us-aspire-0.2",
    "fr": "C:\\Users\\jenny\\Notebooks\\vosk-model-small-fr-0.22",
    "es": "C:\\Users\\jenny\\Notebooks\\vosk-model-small-es-0.42",
    "ja": "C:\\Users\\jenny\\Notebooks\\vosk-model-small-ja-0.22",
    "zh-CN": "C:\\Users\\jenny\\Notebooks\\vosk-model-small-cn-0.22"
}
```

```
# Setup our language recognizer with our models
def setup_recognizer(language):
    model_name = vosk_model_names[language]
    model = vosk.Model(model_name)
    recognizer = vosk.KaldiRecognizer(model, 16000)
    recognizer.SetWords(True)
    return recognizer

rec = setup_recognizer("en")

record_button = widgets.Button(description="Record")
stop_button = widgets.Button(description="Stop")


# Define the audio stream callback function
def callback(in_data, frame_count, time_info, status):
    if status:
        print(status, file=sys.stderr)
    # Convert the audio data to the format expected by Vosk
    if in_data:
        if len(in_data) % 2 == 1:
            in_data = in_data[:-1]
        rec.AcceptWaveform(in_data)
    return (in_data, pyaudio.paContinue)

# Define the PyAudio interface
p = pyaudio.PyAudio()

# event handling for the recording and stopping buttons when clicked by user
# Recording on function
def on_record_clicked(b):
    global stream, rec
    rec = setup_recognizer(recognition_dropdown.value)
    # Ref. test_microphone.py from skabbit on github
    stream = p.open(format=pyaudio.paInt16, channels=1, rate=16000, input=True,
frames_per_buffer=16000, stream_callback=callback)
    stream.start_stream()

# Recording stop function
def on_stop_clicked(b):
    global p, stream, translated_text
    stream.stop_stream()
    stream.close()
    p.terminate()
    text = rec.FinalResult()
```

```
    result = json.loads(text)
    print("Result:", result["text"])

    if result["text"]:
        try:
            translated_text = translate_text(result["text"], dropdown.value)
            print(f"Translated text ({dropdown.value}): {translated_text}")
        except AttributeError as e:
            print("Error:", str(e))

# Recording on or stopped function calls
record_button.on_click(on_record_clicked)
stop_button.on_click(on_stop_clicked)

# Define the translation function
def translate_text(text, target_language):
    try:
        translator = Translator(to_lang=target_language)
        translation = translator.translate(text)
        return translation
    except Exception as e:
        print("Error:", str(e))
        return ""

# Define the GUI elements
dropdown = widgets.Dropdown(options=[("English", "en"), ("French", "fr"), ("Spanish", "es"),
("Japanese", "ja"), ("Chinese", "zh-CN")], description="Language:")
output_box = widgets.Output()
play_button = widgets.Button(description="Play Translation")
audio_output = widgets.Output()

# Define the click event function for the play button
def on_play_button_clicked(b):
    with audio_output:
        global translated_text
        # If the user want to play transcription audio file then play file with error handling
        if translated_text:
            tts = gTTS(text=translated_text, lang=dropdown.value)
            with io.BytesIO() as f:
                tts.write_to_fp(f)
                f.seek(0)
                display(Audio(f.read(), autoplay=True))
        else:
            #User needs to hit translate before playing the file
            print("Please translate the text first.")
```

```
# Speech recognition language selection with a dropdown menu
recognition_dropdown = widgets.Dropdown(options=[("English", "en"), ("French", "fr"),
("Spanish", "es"), ("Japanese", "ja"), ("Chinese", "zh-CN")], description="Recognize:")
display(recognition_dropdown)

# Text-to-speech setup
tts_engine = pyttsx3.init()
voices = tts_engine.getProperty('voices')
voice_dropdown = widgets.Dropdown(options=[(voice.name, voice.id) for voice in voices],
description="Voice:")
display(voice_dropdown)

# attaches the event function to the play button
play_button.on_click(on_play_button_clicked)

# layout for widgets
input_widgets = widgets.VBox([widgets.HBox([recognition_dropdown, dropdown,
voice_dropdown]), widgets.HBox([record_button, stop_button, play_button])])
display(input_widgets)
display(output_box)
display(audio_output)
```

```
            print("Please translate the text first.")

# Speech recognition language selection
recognition_dropdown = widgets.Dropdown(options=[("English", "en"), ("French", "fr")
display(recognition_dropdown)

# Text-to-speech engine setup
tts_engine = pyttsx3.init()
voices = tts_engine.getProperty('voices')
voice_dropdown = widgets.Dropdown(options=[(voice.name, voice.id) for voice in voice
display(voice_dropdown)

# Attach the event function to the play button
play_button.on_click(on_play_button_clicked)

# Improved layout
input_widgets = widgets.VBox([widgets.HBox([recognition_dropdown, dropdown, voice_dr
display(input_widgets)
display(output_box)
display(audio_output)
```




```
Result: you stink
Translated text (ja): 君は臭いよ
```

Program output:
Program Demo Youtube Link:
https://youtu.be/yx2j4mOVYXA

Conclusion

Our biggest flaw in our project was experimenting with our variables and having a concrete result to finalize our research. The use of complex algorithms and deep learning techniques like Mel-frequency cepstral coefficients(MFCCs), Convolutional neural networks(CNNs), Recurrent neural networks (RNNs), and Connectionist temporal classification (CTC) loss can be challenging to implement and tune, especially when our team has limited experience with such complicated process. In the end, we felt that our work was insufficient and incomplete as we were unsatisfied by our results and lack of knowledge. We wanted to be able to combine our acoustic model with a language model and a decoding algorithm but felt that we were inadequate in skills and setting up our environment correctly so we didn't finish in time. We likely approached the project the wrong way and needed to think of a better way to structure it. We hope to learn more in the future and adapt a more sophisticated ASR Translation Program.

References

Alphacep. “Test_microphone.Py Has OSError: [Errno -9981] Input Overflowed on OSX · Issue #128 · Alphacep/Vosk-Api.” GitHub. N.p., n.d. Web. 12 May 2023.

Chung, Hoon et al. “Semi-Supervised Speech Recognition Acoustic Model Training Using Policy Gradient.” MDPI. Multidisciplinary Digital Publishing Institute, 20 May 2020. Web. 12 May 2023.

Graves, Alex, Abdel-rahman Mohamed, and Geoffrey Hinton. “Speech Recognition with Deep Recurrent Neural Networks.” arXiv.org. N.p., 22 Mar. 2013. Web. 12 May 2023.

Hannun, Awni. “Sequence Modeling with CTC.” Distill. N.p., 3 Jan. 2020. Web. 12 May 2023.

Hochreiter, Sepp, and Jürgen Schmidhuber. Long Short Term Memory. N.p., 1995. Web.

Nandi, Papia. “CNNs for Audio Classification.” Medium. Towards Data Science, 10 Dec. 2021. Web. 12 May 2023.

“Simple Audio Recognition: Recognizing Keywords : TensorFlow Core.” TensorFlow. N.p., n.d. Web. 12 May 2023.

SociallyIneptWeeb. “SociallyIneptWeeb/LanguageLeapAI: Your Personal Multilingual AI Translator.” GitHub. N.p., n.d. Web. 12 May 2023.

“Speech Command Classification with Torchaudio¶.” Speech Command Classification with torchaudio - PyTorch Tutorials 1.13.1+cu117 documentation. N.p., n.d. Web. 12 May 2023.

Team, Keras. “Keras Documentation: Automatic Speech Recognition Using CTC.” Keras. N.p., n.d. Web. 12 May 2023.

---. “Keras Documentation: The Model Class.” Keras. N.p., n.d. Web. 12 May 2023.

“Tf.Keras.Layers.Activation : TensorFlow v2.12.0.” TensorFlow. N.p., n.d. Web. 12 May 2023.

“Tf.Keras.Layers.BatchNormalization : TensorFlow v2.12.0.” TensorFlow. N.p., n.d. Web. 12 May 2023.

“Tf.Keras.Layers.Bidirectional : TensorFlow v2.12.0.” TensorFlow. N.p., n.d. Web. 12 May 2023.

“Tf.Keras.Layers.Conv2D : TensorFlow v2.12.0.” TensorFlow. N.p., n.d. Web. 12 May 2023.

“Tf.Keras.Layers.Dense : TensorFlow v2.12.0.” TensorFlow. N.p., n.d. Web. 12 May 2023.

“Tf.Keras.Layers.Dropout : TensorFlow v2.12.0.” TensorFlow. N.p., n.d. Web. 12 May 2023.

“Tf.Keras.Layers.Flatten : TensorFlow v2.12.0.” TensorFlow. N.p., n.d. Web. 12 May 2023.

“Tf.Keras.Layers.Lambda : TensorFlow v2.12.0.” TensorFlow. N.p., n.d. Web. 12 May 2023.

“Tf.Keras.Layers.LSTM : TensorFlow v2.12.0.” TensorFlow. N.p., n.d. Web. 12 May 2023.

“Tf.Keras.Layers.ReLU : TensorFlow v2.12.0.” TensorFlow. N.p., n.d. Web. 12 May 2023.

“Tf.Keras.Layers.RNN : TensorFlow v2.12.0.” TensorFlow. N.p., n.d. Web. 12 May 2023.

“Tf.Keras.Layers.TimeDistributed : TensorFlow v2.12.0.” TensorFlow. N.p., n.d. Web. 12 May 2023.

“Tf.Keras.Optimizers.Adam : TensorFlow v2.12.0.” TensorFlow. N.p., n.d. Web. 12 May 2023.

Zen, Heiga. “Acoustic Modeling for Speech Synthesis: From HMM to RNN.” Google Research. N.p., 1 Jan. 1970. Web. 12 May 2023.

Lecture References Sources

Konopka, Chuck. "An Introduction to Discrete Hidden Markov Model." CS 582: Intro to Speech Processing, San Diego State University, Lectures 8-9.

Konopka, Chuck. " Speech Synthesis." CS 582: Intro to Speech Processing, San Diego State University.

Xu, Yang. “Assignment 10 (Part 2): Character-Level RNN-Based Text Classification Model” CS-549 Machine Learning, San Diego State University, 2022.

Xu, Yang. “Assignment 8: Optimization of Deep Neural Networks” CS-549 Machine Learning, San Diego State University, 2022.

Xu, Yang. “Computational Graph and Neural Network - activation/cost.” CS549: Machine Learning. San Diego State University.

Xu, Yang. “Convolutional Neural Network(CNN).” CS549: Machine Learning. San Diego State University.

Xu, Yang. “Generative Adversarial Networks (GANs).” CS549: Machine Learning. San Diego State University.

Xu, Yang. “Optimization of Training Deep Neural Networks(DNN) - ReLU .” CS549: Machine Learning. San Diego State University.

Xu, Yang. “Recurrent Neural Network(RNN).” CS549: Machine Learning. San Diego State University.

Xu, Yang. “Transformer and Transformer-based Models.” CS549: Machine Learning. San Diego State University.

Xu, Yang. “Word Embedding - LSTM.” CS549: Machine Learning. San Diego State University.