Speaker Recognition in Natural Language Processing: A Comprehensive Overview

https://medium.com/the-modern-scientist/speaker-recognition-in-natural-language-processing-a-comprehensive-overview-03c3138b5e52

Introduction

In the realm of Natural Language Processing (NLP), speaker recognition is a critical subfield that focuses on identifying and verifying the identity of a speaker based on their vocal characteristics. While NLP primarily deals with understanding and processing text, the integration of speaker recognition techniques broadens its scope and applicability. This essay delves into the concept of speaker recognition in NLP, discussing its significance, methods, challenges, and applications.

Speaker Recognition in Natural Language Processing: Unveiling the Voice of Identity.

Significance of Speaker Recognition in NLP

Speaker recognition is a powerful tool that enhances NLP systems in various ways. Its significance can be understood through the following key points:

1. **Security and Access Control:** In today's digital world, securing access to sensitive information is a paramount concern. Speaker recognition is commonly used in applications such as voice-activated

authentication systems, allowing users to access their accounts and devices securely.

2. **Personalized User Experiences**: Many NLP applications benefit from recognizing a user's voice. For instance, voice assistants like Siri and Alexa can personalize responses based on the speaker's preferences and previous interactions.

3. **Forensic Analysis:** Speaker recognition plays a pivotal role in forensic analysis, aiding law enforcement agencies in identifying criminals through intercepted voice communications.

4. **Customer Service:** In the customer service industry, recognizing individual callers can lead to a more personalized and efficient service experience. Automated call centers can use speaker recognition to route calls to the most relevant agents.

Methods of Speaker Recognition

Speaker recognition can be broadly classified into two categories: speaker identification and speaker verification.

1. **Speaker Identification**: This process involves determining the identity of a speaker from a set of known speakers. It typically employs techniques such as Gaussian Mixture Models (GMM), Hidden Markov Models (HMM), and deep learning methods like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). These models analyze acoustic features, such as Mel-frequency cepstral coefficients (MFCCs), to distinguish between different speakers.

2. **Speaker Verification:** Speaker verification, on the other hand, focuses on verifying whether a given voice sample matches a particular speaker's identity. This is commonly used in applications requiring authentication, where the system compares the voice input with a stored voiceprint.

Challenges in Speaker Recognition

Despite the potential of speaker recognition in NLP, it faces several challenges:

1. **Variability in Voice:** Speakers' voices can vary significantly due to factors such as emotional state, health, and environmental conditions. Robust speaker recognition systems must account for these variations.
2. **Data Privacy:** The collection and storage of voice data for recognition purposes raise concerns about privacy and data security. Striking a balance between utility and user privacy is a complex challenge.
3. **Speaker Impersonation:** Malicious actors can attempt to impersonate legitimate speakers, making it essential for recognition systems to be robust against such attacks.
4. **Limited Data:** Developing accurate speaker recognition models requires substantial labeled data for training. This can be a limitation, especially for less-represented languages and dialects.

Applications of Speaker Recognition in NLP

Speaker recognition in NLP has found its way into a wide range of applications, including:

1. **Voice Assistants:** Popular voice assistants like Siri, Google Assistant, and Amazon's Alexa use speaker recognition to personalize responses and identify different users in multi-user households.
2. **Banking and Finance:** Speaker recognition is used to secure phone banking and access to financial services, adding an extra layer of authentication.
3. **Law Enforcement:** In the realm of criminal investigations, speaker recognition helps law enforcement agencies identify suspects through voice analysis.
4. **Healthcare:** In telemedicine and healthcare, speaker recognition can be used to authenticate doctors and patients for secure and confidential communication.

Code

Creating a complete Python code for speaker recognition with plots is a complex task that would require extensive libraries, data, and time. However, I can provide you with a simplified example using a pre-trained model, libraries for audio processing, and some basic plots.

In this example, we'll use the `pyAudioAnalysis` library for feature extraction, and `scikit-learn` for classification. Note that for a production-level speaker recognition system, you would require a substantial amount of labeled audio data and more sophisticated models.

```
# Import necessary libraries
import pyaudio
import wave
import os
import numpy as np
import matplotlib.pyplot as plt
from pyAudioAnalysis import audioBasicIO
from pyAudioAnalysis import audioFeatureExtraction
```

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler

# Define functions for audio feature extraction
def extract_features(file_path):
    [Fs, x] = audioBasicIO.read_audio_file(file_path)
    F, f_names = audioFeatureExtraction.short_term_feature_extraction(x, Fs, 0.050*Fs,
0.025*Fs)
    return F

# Create a dataset of audio features (sample data)
dataset_path = "speaker_data"
speakers = ["speaker1", "speaker2"]
X = []
y = []

for speaker in speakers:
    speaker_folder = os.path.join(dataset_path, speaker)
    for audio_file in os.listdir(speaker_folder):
        if audio_file.endswith(".wav"):
            feature_vector = extract_features(os.path.join(speaker_folder, audio_file))
            X.append(feature_vector)
            y.append(speaker)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train a classifier (Support Vector Machine)
clf = SVC()
clf.fit(X_train, y_train)

# Predict and evaluate
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy*100:.2f}%")

# Plot the results
unique_speakers = np.unique(speakers)
confusion_matrix = np.zeros((len(unique_speakers), len(unique_speakers))
for true, pred in zip(y_test, y_pred):
    confusion_matrix[np.where(unique_speakers == true), np.where(unique_speakers ==
pred)] += 1

plt.imshow(confusion_matrix, interpolation="nearest", cmap=plt.cm.Blues)
plt.title("Speaker Recognition Confusion Matrix")
plt.colorbar()

tick_marks = np.arange(len(unique_speakers))
plt.xticks(tick_marks, unique_speakers, rotation=45)
plt.yticks(tick_marks, unique_speakers)

plt.tight_layout()
plt.ylabel("True label")
```

```
    plt.xlabel("Predicted label")
    plt.show()
```

In this code:

1. We use `pyAudioAnalysis` for audio feature extraction, `scikit-learn` for machine learning, and `matplotlib` for plotting.
2. The code assumes you have audio data in the “speaker_data” folder, where subfolders contain audio files for different speakers.
3. Features are extracted from the audio files and used to train a Support Vector Machine (SVM) classifier.
4. The accuracy of the classifier is calculated and a confusion matrix is plotted.

Remember, this is a simplified example. In a real-world scenario, you would need a more extensive dataset and possibly more complex models for robust speaker recognition.

Conclusion

Speaker recognition in NLP is a dynamic and evolving field with diverse applications and substantial potential. As technology continues to advance, the accuracy and reliability of speaker recognition systems will improve, addressing many of the existing challenges. The integration of speaker recognition techniques into NLP not only enhances user experiences but also adds a layer of security and personalization, making it a valuable asset in the modern digital landscape. However, it is imperative to address privacy concerns and ethical considerations to ensure responsible and secure implementation in the ever-expanding world of NLP applications.

Speaker Recognition: Unlocking the Power of Voice

https://medium.com/the-modern-scientist/speaker-recognition-unlocking-the-power-of-voice-d59c40db5450

Introduction

In our ever-evolving world of technology, voice-based interactions have become increasingly prevalent. From virtual assistants to voice-controlled devices, the ability to recognize and authenticate individuals based on their unique vocal characteristics has gained significant importance. Speaker recognition, a subfield of biometrics, offers a promising solution by leveraging the distinct patterns present in an individual's voice to identify and verify their identity. This essay explores the fundamentals, applications, challenges, and advancements in speaker recognition, shedding light on its growing significance in our modern society.

Understanding Speaker Recognition

Speaker recognition, also known as voice recognition or speaker identification, is the process of identifying and verifying the identity of a speaker based on their unique vocal characteristics. These characteristics encompass a wide range of factors, including pitch, accent, intonation, speech patterns, and pronunciation nuances. By analyzing these distinct features, sophisticated algorithms and models can determine the likelihood of a speaker's identity, comparing it with stored voice profiles in a database.

Applications of Speaker Recognition

1. **Forensic Investigations**: Speaker recognition plays a vital role in law enforcement and forensic investigations. It enables the identification of individuals based on recorded voice samples, aiding in the resolution of criminal cases and providing crucial evidence in court proceedings.

2. **Access Control and Security**: Speaker recognition has found significant application in access control systems, enhancing security measures in various domains. Voice-based authentication can provide secure and convenient access to restricted areas, devices, or accounts, replacing traditional methods such as PINs or passwords.

3. **Telecommunications and Customer Service**: Speaker recognition technology is employed in telecommunication systems to authenticate users during phone-based transactions, ensuring secure and convenient interactions. Additionally, it assists in providing personalized customer service experiences, enabling automated systems to recognize and respond to individual callers.

4. **Voice Assistants and Home Automation**: Virtual assistants like Siri, Alexa, and Google Assistant rely on speaker recognition to differentiate between different users within a household. This allows for personalized responses, tailored recommendations, and customized user experiences.

Challenges in Speaker Recognition

Despite the advancements in speaker recognition technology, several challenges persist, posing limitations and room for improvement:

1. **Variability in Voice Data**: Factors such as background noise, microphone quality, and emotional state can affect the quality and consistency of voice data, making accurate recognition more challenging.

2. **Impersonation and Spoofing**: The vulnerability of speaker recognition systems to impersonation and spoofing poses a significant challenge. Adversaries may attempt to mimic or manipulate voice samples to gain unauthorized access or deceive the system, necessitating robust anti-spoofing techniques.

3. **Privacy and Ethical Considerations**: The collection and storage of voice data raises concerns regarding privacy, security, and ethical use. Striking a balance between the convenience of voice-based authentication and safeguarding individuals' personal information is crucial.

Advancements in Speaker Recognition:

Researchers and technologists continue to make remarkable progress in the field of speaker recognition. Recent advancements include:

1. **Deep Learning and Neural Networks**: The adoption of deep learning techniques, particularly neural networks, has significantly improved the accuracy and robustness of speaker recognition systems. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have shown promising results in voice feature extraction and modeling.

2. **Multi-Modal Approaches**: Integrating multiple modalities, such as speech and visual cues, can enhance speaker recognition systems' performance and security. Combining audio analysis with lip movement, facial recognition, or behavioral patterns provides a more comprehensive and reliable means of speaker identification.

3. **Anti-Spoofing Measures**: Researchers are actively developing and refining anti-spoofing techniques to counter fraudulent attempts to deceive speaker recognition systems. These measures involve analyzing various aspects of voice data, such as high-frequency components, acoustic properties, and temporal characteristics, to detect spoofing attacks.

There are several techniques and approaches used in speaker recognition systems. Here are some commonly employed techniques:

1. **Feature Extraction**: Feature extraction is a crucial step in speaker recognition, where relevant information is extracted from speech signals to represent the speaker's characteristics. Some commonly used features include:
   - Mel-Frequency Cepstral Coefficients (MFCCs): These coefficients represent the spectral envelope of the speech signal, capturing information about the shape of the vocal tract.
   - Linear Predictive Coding (LPC): LPC analyzes the linear prediction error of the speech signal, capturing information about the vocal tract resonances.
   - Perceptual Linear Prediction (PLP): PLP combines aspects of MFCC and LPC techniques, considering both the spectral and temporal characteristics of the speech signal.

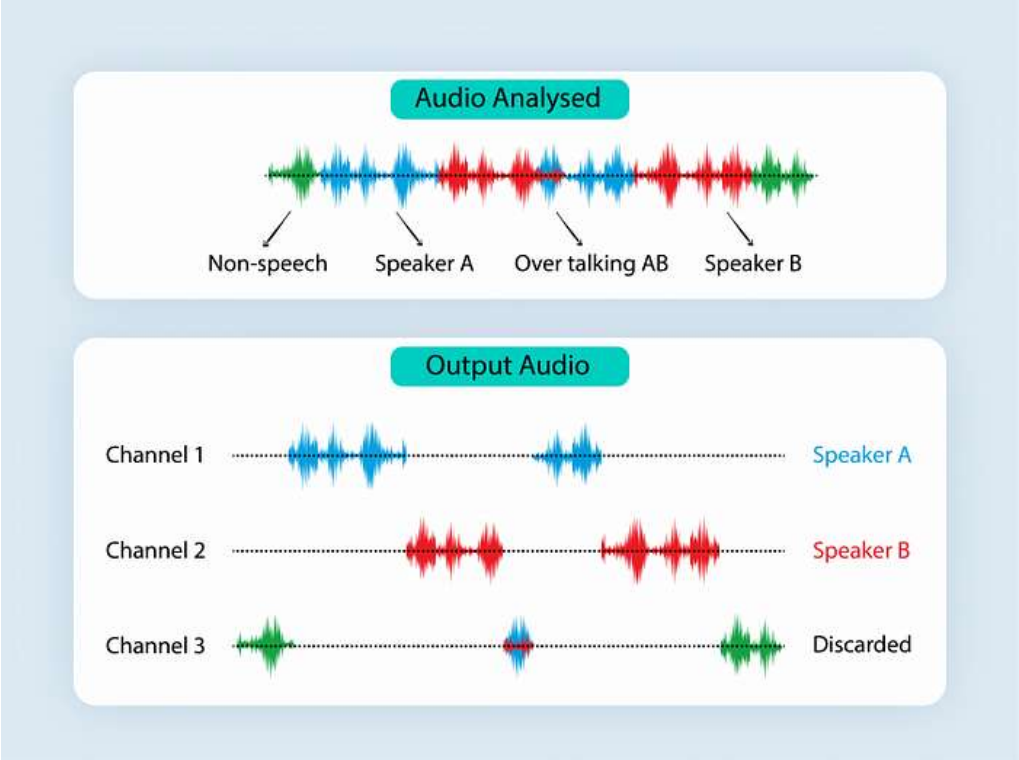
2. **Speaker Modeling:** Once the features are extracted, various modeling techniques are employed to represent the speaker's characteristics. Some common modeling approaches include:
   - Gaussian Mixture Models (GMMs): GMMs are probabilistic models that represent the statistical distribution of speaker-specific feature vectors. They can be trained to estimate the likelihood of a given feature vector belonging to a particular speaker.
   - Hidden Markov Models (HMMs): HMMs are widely used for speech and speaker recognition. They model the temporal dynamics of speech and capture the transitions between different speech sounds or speaker characteristics.
   - Support Vector Machines (SVMs): SVMs are supervised machine learning models that can be trained to classify speaker-specific feature vectors based on a given training set.
   - Deep Neural Networks (DNNs): DNNs, particularly Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have shown

promising results in speaker recognition. They can learn complex representations from raw audio data and capture both spectral and temporal information effectively.

3. **Enrollment and Verification:** The speaker recognition process typically involves two main steps: enrollment and verification.
   - Enrollment: During enrollment, the system creates a speaker model or template by training the chosen modeling technique on a set of known or labeled speaker data. This template represents the unique characteristics of the speaker's voice.
   - Verification: In the verification phase, the system compares a test sample to the enrolled speaker models. The similarity or distance between the test sample and each enrolled model is computed, and a decision is made based on a predefined threshold to accept or reject the claimed speaker's identity.

4. **Anti-Spoofing Techniques:** To mitigate the risk of spoofing attacks and ensure the integrity of the speaker recognition system, various anti-spoofing techniques are employed. These techniques aim to differentiate between genuine speech and artificially generated or manipulated speech samples. Common anti-spoofing methods include analyzing high-frequency components, detecting voice activity, examining acoustic properties, and employing machine learning algorithms to identify spoofed or manipulated samples.

It's important to note that the choice of techniques and algorithms may vary depending on the specific requirements, dataset availability, and the complexity of the speaker recognition task. Researchers and practitioners continue to explore new techniques and combine multiple approaches to improve the accuracy, robustness, and security of speaker recognition systems.



Speaker recognition has made significant progress over the years, but there are still several open problems and challenges that researchers and technologists are actively addressing. Some of the key open problems in speaker recognition include:

1. **Robustness to Variability:** Speaker recognition systems often struggle with handling variability in speech, including different speaking styles, accents, languages, and emotional states. Developing models and algorithms that can effectively handle such variability and provide accurate recognition regardless of these factors remains an open problem.

2. **Speaker Diarization:** Speaker diarization involves segmenting an audio recording into individual speaker segments. It is a crucial step in speaker recognition systems, especially in scenarios where multiple speakers are present. Accurate and efficient diarization techniques that can handle overlapping speech, background noise, and speaker turn-taking in real-world environments are areas of active research.

3. **Data Scarcity and Diversity:** The availability of large and diverse speaker datasets plays a vital role in training robust speaker recognition models. However, acquiring such datasets can be challenging due to privacy concerns, especially when dealing with sensitive voice data. Developing techniques to overcome data scarcity while ensuring data privacy and diversity remains an open problem.

4. **Cross-lingual and Cross-domain Recognition:** Many speaker recognition systems are designed and trained on specific languages or domains, limiting their effectiveness in cross-lingual or cross-domain scenarios. Developing techniques that can generalize well across different languages, dialects, and domains is an ongoing challenge in the field.

5. **Vulnerability to Adversarial Attacks:** Speaker recognition systems are susceptible to adversarial attacks, where an adversary deliberately manipulates the voice samples to deceive the system. Adversarial attacks can include impersonation, voice synthesis, or modifying audio signals to alter the recognized speaker's identity. Developing robust anti-spoofing techniques and ensuring system security against such attacks is a critical open problem.

6. **Privacy and Ethical Considerations:** As speaker recognition technology becomes more prevalent, concerns around privacy and ethical use of voice data are increasing. Designing systems that prioritize user privacy, obtain informed consent, and implement secure data storage and handling mechanisms are ongoing challenges to address.

7. **Real-time and Resource-constrained Applications:** Speaker recognition systems are often required to operate in real-time or on resource-constrained devices, such as smartphones or IoT devices. Ensuring efficient and accurate speaker recognition in these scenarios, where computational resources and processing power are limited, is an open problem.

Addressing these open problems in speaker recognition requires interdisciplinary research, encompassing areas such as signal processing, machine learning, natural language processing, and human-computer interaction. Continued collaboration and innovation in these fields will contribute to the development of more robust, accurate, and secure speaker recognition systems in the future.

Here's an example of speaker recognition code in Python using the scikit-learn library and the Gaussian Mixture Model (GMM) approach for modeling:

```
import numpy as np
from sklearn.mixture import GaussianMixture

# Training data
# Each row represents the feature vector of a speaker
train_data = np.array([
    [0.1, 0.2, 0.3, 0.4],  # Speaker 1
    [0.2, 0.3, 0.4, 0.5],  # Speaker 1
    [0.9, 0.8, 0.7, 0.6],  # Speaker 2
    [0.8, 0.7, 0.6, 0.5]   # Speaker 2
])

# Create labels for the training data
train_labels = np.array([0, 0, 1, 1])  # 0 represents Speaker 1, 1 represents Speaker 2

# Testing data
# Each row represents the feature vector of a test sample
test_data = np.array([
    [0.3, 0.4, 0.5, 0.6],  # Unknown speaker
    [0.7, 0.6, 0.5, 0.4]   # Unknown speaker
])

# Train the Gaussian Mixture Model (GMM) with the training data
gmm = GaussianMixture(n_components=2)  # Number of components equals the number of
speakers
gmm.fit(train_data)

# Predict the labels for the testing data
predicted_labels = gmm.predict(test_data)

# Display the predicted labels
for label in predicted_labels:
    print("Predicted Speaker:", label)
```

In this example, we have two speakers represented by their respective feature vectors in the `train_data` array. The corresponding labels are provided in

the `train_labels` array. We then create a GMM object with two components (representing the two speakers) using `GaussianMixture` from scikit-learn. The GMM is trained on the training data using the `fit()` method.

Next, we have some test samples represented by feature vectors in the `test_data` array. We use the trained GMM model to predict the labels for these test samples using the `predict()` method. The predicted labels are stored in the `predicted_labels` array.

Finally, we display the predicted labels to identify the corresponding speakers.

Note: This is a simplified example for illustration purposes. In practice, you may need to preprocess the audio data, extract appropriate features (such as MFCCs), and handle larger datasets. Additionally, consider incorporating anti-spoofing techniques and other enhancements for a more robust speaker recognition system.

Conclusion

Speaker recognition has emerged as a powerful technology with a wide range of applications in various sectors, including security, telecommunications, and personalization. While significant progress has been made, there are still challenges to overcome, such as variability in voice data and the potential for spoofing. Nonetheless, ongoing advancements in deep learning, multi-modal approaches, and anti-spoofing techniques offer promising solutions. As the field continues to evolve, speaker recognition is poised to play an increasingly integral role in our voice-driven future, enabling secure and personalized interactions with technology.

Speaker Recognition

Model Building

https://medium.com/@makcedward/speaker-recognition-c133ed89ad7c

1. Collect a dataset of audio recordings: You will need a dataset of audio recordings of people speaking to train your speaker recognition model. This dataset should include multiple recordings of each speaker. You may use recordings from a variety of sources, such as public datasets (e.g. VoxCeleb), private datasets, or a combination of both.

2. Extract features from the audio recordings: Once you have collected your audio recordings, you need to extract features from them. Common features used for speaker recognition include Mel-Frequency Cepstral Coefficients (MFCCs), Linear Predictive Coding (LPC) coefficients, and other speech processing techniques.

3. Train a model on the extracted features: After extracting the features from the audio recordings, you can use them to train a model for speaker recognition. Popular models for speaker recognition include Hidden Markov Models (HMMs), and Deep Neural Networks (DNNs).

4. Test the model: Once you have trained your model, you can test it on unseen audio recordings to measure its performance via an equal error rate (EER). This can be done by comparing the model's predictions against the true speaker labels in the test dataset.

Speaker Recognition from Audio

https://medium.com/@makcedward/speaker-recognition-from-audio-956e24052af0

Photo by Priscilla Du Preez on Unsplash

In my previous work, I focused on text-based machine learning (ML) models such as named entity recognition (NER), intention classification, and topic modeling. I am preparing a new series of blogs that are acoustic-related topics. This is the first post of this series, so I will try to illustrate the overall landscape of the acoustic domain via one of the classific problems.

Data Variety

Photo by Luke Michael on Unsplash

Voice includes lots of variety even though people speak the same transcript. You can distinguish two voices who speak from two different people. Even for the same person, voices can be different when you have different emotions (e.g., happy, angry, sad, etc.).

Input Type

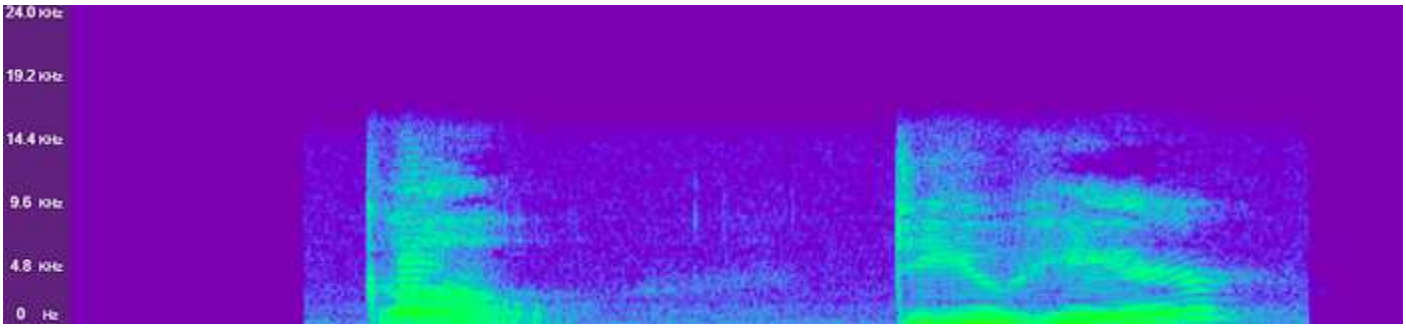
Photo by Marius Masalar on Unsplash

We may feed bytes, characters, or tokens into a text-based ML model. On the other hand, we can feed waveform or spectrogram into an acoustic ML model. A spectrogram is a visualization of representing the signal (i.e., voice) **strength** (i.e., loudness) of a signal over **time** at various **frequencies**.

Waveform Visualization

In exploratory data analysis, it is not that hard to explore characters or tokens. Unlike text, we can not visualize audio clips directly. Waveform and spectrogram visualization is a good way to understand our data.

You may use audio-preview plugin (if you are using Visual Studio), native software on your machine, or Jupyter Notebook (with librosa) to understand it. Alternatively, we may use cloud solutions to achieve the same purpose. For example, once uploading files to DagsHub, everyone with access is able to listen and visualize it via browser.



Spectrogram Visualization

Lots of modern ML models consume spectrograms (or mel-spectrograms) instead of waveforms for several reasons. Waveform includes more information but spectrogram are closed to the human auditory system. Another reason is that we can reuse computer vision model architecture on the acoustic models. I will show how to use the computer vision (CV) architecture, ResNet [1], and speaker recognition model later.

Speaker Recognition

Photo by Kevin Ku on Unsplash

Our toy problem is speaker recognition. Given voice input, we want to identify the speaker. It is similar to facial recognition and finger recognition when you try to unlock your iPhone.

Both speaker verification and identification are under the speaker recognition umbrella. Speaker identification refers to determining who the enrolled speaker is. Speaker verification means either accepting or rejecting the identity claimed by a speaker.

Dataset

Photo by Tobias Fischer on Unsplash

In the regression problem, we have the titanic dataset. In audio problems, I usually start from the LibriSpeech dataset [2]. You may download it from OpenSLR, PyTorch, TensorFlow, or HuggingFace.

Although we can download it from the internet easily, it is raw audio data (i.e., waveform). As mentioned in the previous section, we feed spectrogram data to ML models instead of waveform data. You may convert waveform data to spectrogram and persist in your local machine. It is fine if you work alone without

any scale requirement. Another option is uploading it to a cloud provider (e.g., AWS, GCP), but you have to manage it by yourself.

Alternatively, you may consider using the Direct Data Access feature which is provided by DagsHub. It helps us to streamline the process of uploading and downloading from the cloud. We can focus on model training rather than infrastructure.

The following codes show how to upload files to DagsHub's cloud.

```
# Initialize remote date repository
from dagshub.upload import Repo
repo = Repo(repo_owner, repo_name)
dataset_dir = 'datastore'
ds = repo.directory(dataset_dir)

# Iterlate local audio file paths
for f in audio_file_paths:
    file_name = os.path.basename(f)
    _id, session, _ = file_name.split('-')

    remote_file_path = os.path.join(
        'LibriSpeech',
        _id,
        session,
        file_name
    )

    # Upload file
    ds.add(file=f, path=remote_file_path)

# Commit change on files upload
ds.commit(f"Upload {total_cnt} audio files", versioning="dvc")
```

We simply use the API from DagsHub to get back data. No major difference when loading files locally.

```
from dagshub.streaming import DagsHubFilesystem
fs = DagsHubFilesystem()

import csv

metadata = {}
```

```
with fs.open('datastore/LibriSpeech/metadata.csv') as infile:
    reader = list(csv.reader(infile))
    for row in reader[1:]:
        metadata[row[0]] = len(metadata)
```

After talking about data storage, we will move to data processing. Converting waveform to mel-spectrogram is a very typical process, and most of the models consume it. Therefore, you may consider caching the processed result.

Data Processing

Photo by NEW DATA SERVICES on Unsplash

librosa [3] is a famous python package for music and audio analysis. It provides an easy way to convert raw waveform data to mel-spectrogram data.

```
# Load audio data
y, sr = librosa.load(file_path, sr=sample_rate)
# Convert waveform to mel-spectogram
spec=librosa.feature.melspectrogram(y=f.numpy(), sr=sample_rate)
# Convert a power spectrogram (amplitude squared) to decibel (dB) units
spec_db=librosa.power_to_db(spec)
```

Sample rate means the number of samples per second. For example, 16000 means there are 16000 data points per second. Higher sample rates include more data points. 16000, 22050, 44100, and 48000 are sample rates that I usually use.

When working on audio data, it will be good to have the same sample rate across all data feeding into the ML model.
Passing *sample_rate* into *librosa.load* function, which helps to convert the audio clip to the expected sample rate.

Modeling

Photo by Kelly Sikkema on Unsplash

I mentioned we could leverage CV model architecture on the acoustic model. Simply load the ResNet34 model by using torchvision package, we have the pre-trained model now.

```
from torchvision.models import resnet34
import torch.nn as nn
# Load ResNet34 CV model
resnet_model = resnet34(weights=True)
# Adjusted the last layer output for our binary classification use case
resnet_model.fc = nn.Linear(512, label_cnt)
# Adjust the first convolution neural network (CNN) layer for our use case
resnet_model.conv1 = nn.Conv2d(
  1,
  64,
  kernel_size=(7, 7),
  stride=(2, 2),
  padding=(3, 3),
  bias=False
)
```

Tracking

Photo by Sandra Tan on Unsplash

Experiment tracking is very important when building a model. It allows us to compare model metrics (performance, training time, etc) against different settings. You can mark it down into a spreadsheet or use experiment tracking tools. Weights & Biases, Comet ML, mlflow are some of the great tools that we can use.

We do not need extra tools if we use DagsHub as you just need to run a few lines of code (auto-tracking is available for some frameworks such as PyTorch Lighting) to create the experiment tracking.

```
from dagshub import DAGsHubLogger
logger = DAGsHubLogger(
  metrics_path="logs/test_metrics.csv",
  hparams_path="logs/test_params.yml"
)
logger.log_hyperparams({
    'learning_rate':learning_rate,
    'optimizer': 'adam',
    'epoch': 3,
    'loss': 'ce'
})

accuracy = evaluate(model, x, y_true)
logger.log_metrics({'accuracy': accuracy})
```

Model Registry

Photo by Wedding Dreamz on Unsplash

Besides metrics, we also want to keep the trained model, as we may need to load it later. Model Registry is another important section that we need to take care of. We can simply save the model locally or upload it to the cloud (e.g., AWS, Azure, GCP). However, management is needed but not just storing the file. For example, the association between the experiment and the model is needed. Model access control is suggested as we may only want to share our model with the targeted group of people but not all.

Same to the dataset, we can upload the model to DagsHub so that we have a single view to access the code, experiment, and model.

```
from dagshub.upload import Repo
repo = Repo(REPO_OWNER, REPO_NAME)

dataset_dir = 'model'
ds = repo.directory(dataset_dir)

from dagshub.streaming import DagsHubFilesystem
fs = DagsHubFilesystem()
```

```
ds.add(file='trained_model.pt', path='trained_model.pt')
ds.commit(f"Upload model", versioning="dvc")
```

Take Away

Having a centralized place to keep tracking code, experiment, and model helps us to focus on the model rather than MLOps. A model builder is an expert in training a good model for the company but may not be good at building MLOps infrastructure.

Here is the full script for the model training. To balance the optimization and readability, we simplify the flow such as data streaming, data processing, and modeling. There are a few things that we can further optimize the data processing part. Instead of streamlining data at the very beginning, we should load data on demand to maximize the Direct Data Access feature. Also, we may introduce data augmentation if a more generalized model is needed. To maximize DagsHub Storage feature, we can preprocess (i.e. convert waveform to spectrogram) data and upload it instead of processing data every time.

Here is the repo for the datastore, experiment tracking, and model registry. You may access it to understand how we can combine everything together.

Like to learn?

I am Data Scientist in Bay Area. Focusing on the state-of-the-art in Data Science, Artificial Intelligence, especially in NLP and platform related. Feel free to connect with me on LinkedIn or Github.

Reference

[1] He et al. (2015). Deep Residual Learning for Image Recognition

[2] Panayotov et al. (2015). LibriSpeech: An ASR corpus based on public domain audio books

[3] McFee et al. (2015). librosa: Audio and Music Signal Analysis in Python

Automatic Speaker Recognition using Transfer Learning

https://medium.com/towards-data-science/automatic-speaker-recognition-using-transfer-learning-6fab63e34e74

*When tasked with the challenge of creating a dynamic **voice identifier**, naturally the tool of choice for our team would be an **image classifier***

Even with today's frequent technological breakthroughs in speech-interactive devices (think Siri and Alexa), few companies have tried their hand at enabling multi-user profiles. Google Home has been the most ambitious in this area, allowing up to six user profiles. The recent boom of this technology is what made the potential for this project very exciting to our team. We also wanted to engage in a project that is still a hot topic in deep-learning research, create interesting tools, learn more about neural network architectures, and make original contributions where possible.

We sought to create a system able to quickly add user profiles and accurately identify their voices with very little training data, a few sentences as most! This learning from one to only a few samples is known as *One Shot Learning*. This article will outline the phases of our project in detail.

I. Project Summary

Goal: Classify speakers with minimal training such that only a few words or sentences are needed to achieve high levels of accuracy.

Data: Training data was scraped from Librivox, a source of open domain audiobooks. Testing data was either scraped off YouTube or collected live.

Method: In summary, we converted all of our audio data to spectrogram form. We then trained a CNN derived from Cifar-10 on many speakers as a feature extractor to feed into an SVM for final

classification. This approach is known as *transfer learning*. This approach enabled us to reap the small sample high performance of SVM and feature learning of CNNs.

Applications: The potential applications for our proposed systems are plentiful. They range from home assistant needs (think Alexa and Google Home) to biometric security, marketing tools, and even spying (identifying high profile targets). It could also be used as a tool for speaker diarisation in speech data collection. Given some small previous exposure to included voices, an audio file with multiple speakers could be accurately separated. This opens the door for a lot a more potential "clean data" to be used to create more sophisticated speech-specific models.

Performance: Results were largely positive. With 20–35 seconds of training audio, our model was able to distinguish between three speakers of with 63–95% accuracy in our tests. However, performance drops severely with 5+ speakers or in uniform gender test groups.

Github Link: https://github.com/hamzag95/voice-classification

II. Data Collection

Background

One of the greatest challenges in the field of speaker and speech recognition is the lack of open source data. Most speech data is either proprietary, hard to access, insufficiently labeled, insufficient in amount per speaker, or noisy. In many related research papers, insufficient data has been cited as reasoning for not pursuing further, more complex, models and applications.

We saw this is an opportunity to make a novel contribution to research in this field. Many hours of googling later, our team concluded the best sources of potential audio for our project would be LibriVox, an extensive source of open domain audiobooks, and YouTube. These sources were selected upon best satisfying our criteria shown below.

Audio Source Criteria

- Enough unique speakers for a model to learn generalizable differences in speech
- Male and Female speakers, preferably in a range of languages
- At least 1 hour of audio per speaker available
- Audio can be automatically labeled with meta-data
- Audio has minimal noise (little background noise/music, decent quality, few extraneous sounds)
- Open Domain or Creative Commons License for legal use and data set usage

Scraping Audio from Librivox

We wrote scripts using BeautifulSoup and Selenium to parse the website LibriVox and download the audio books we wanted. BeautifulSoup by itself was not enough, as parts of the website took time (1–2 seconds) to load. Thus we used selenium to wait until certain elements on the webpage appeared and became scrapable.

Our first attempt used a script that moves starts at LibriVox's default home page and downloads all the audio in the page within a certain size file size boundaries. We later realized this was flawed as many audiobooks are actually a collaboration of multiple narrators that would be difficult to automatically

separate. Thus we had to find a way to get audiobooks with unique speakers. Unfortunately, the LibriVox API didn't contain a field to filter by project type (solo or collaboration) or narrator name.

Instead, we used advanced search to include only "solo" narrator books. Soon we realized it was problematic to assume each book had a unique speaker as LibriVox has many repeat narrators. To fix this, we had to read the meta-data of each book to maintain a list of scraped narrators to ensure our data was correctly labeled. In the end, we had 6000+ unique speakers and links to 24000+ hours of audio. However, due to time constraints, we sampled 162 unique speakers for spectrogram conversion. The full list of download links can be found on our project GitHub here.

Scraping Audio from Youtube

From Youtube, we scraped video links from 7 Youtube stars and their tutorial/informative videos. We found that tutorials tended to best fit our standards as they contain mostly clean speech. Selenium was needed to automate the process as scraping YouTube requires scrolling. This process can be seen in real-time in the video below.

Scraping videos on Youtube using Selenium

We didn't scrape more profiles because it was inefficient to manually filter and verify videos based on their inclusion of guest speakers, music, etc...Although tutorial channels generally fit the bill as far as speech cleanliness, they were heavily skewed in gender toward males. Videos would also have varying qualities of audio and background noise. We decided against using what we collected to train the neural net, but thought they would be useful for testing purposes. YouTube remains a source of audio with a lot of potential for data collection but very demanding in terms of dating verification and cleaning.

III. Data Processing

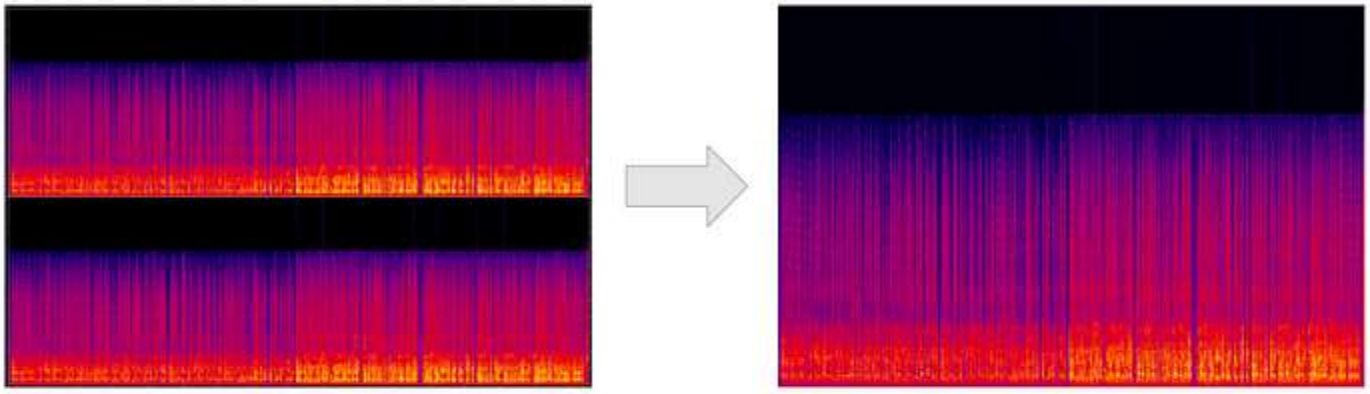
Background

Ultimately, all collected audio had to be converted to 503x800(x3) spectrogram images that captured 5 seconds of audio. The steps for converting the data collected from Librivox and YouTube were slightly different as a result of differing download formats.

For our various processing needs, we were very fortunate to have tools such as ffmpeg, sox, and mp3splt at our disposal that sped up the process while minimizing loss of audio quality.

YoutTube Audio Processing

Once the YouTube video links were collected, we were fortunate to find the YouTube-DL library which allowed us to easily download our desired videos in WAV format. When attempting to convert this data to spectrograms, we came to find that each file was producing two spectrograms because it was stereo audio. This is the major difference we encountered versus processing of LibriVox audio.



Spectrogram from BinarySearchTree.wav stereo file

Spectrogram from BinarySearchTree.wav mono file

Thus, the process can be summarized in the points below:

1. Manually check scraped YouTube Links to verify usability.
2. Download in all verified links in WAV format and automatically label/sort audio

3. Split mono WAV files into 5 seconds segments
4. Convert all stereo WAV files to mono WAV
5. Convert all audio segments to Spectrograms

YouTube data processing phases.

A video of the YouTube audio processing can be seen below:

LibriVox Audio Processing

We processed the LibriVox audio using a single script that placed the data in various levels of processing into different directories so that potential future users could change the segments lengths or conversion types as they please.

The processing can be summarized in the points below:

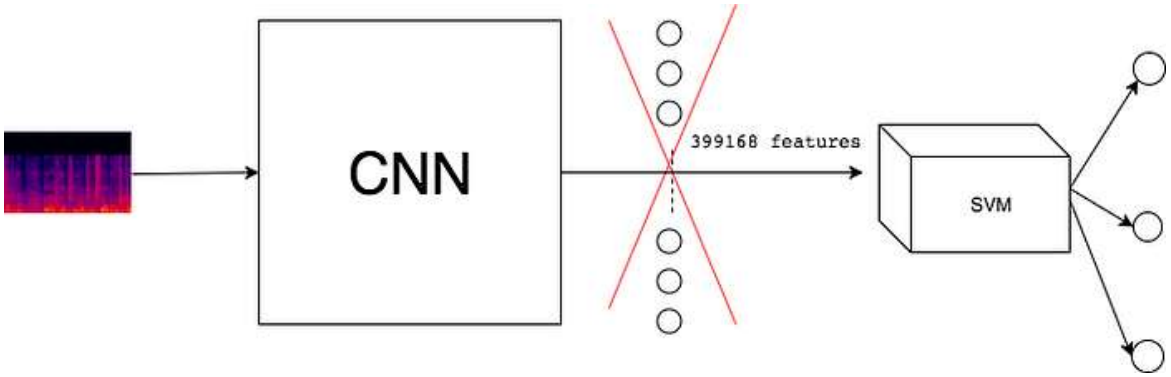
1. Combine all downloaded chapters for a single speaker
2. Trim combined audio to desired length
3. Convert trimmed audio to 16bit 16khz mono WAV using ffmpeg
4. Remove silences longer than .5 seconds
5. Split WAV file in 5 second segments
6. Convert each segment to a spectrogram

LibriVox data processing stages

IV. Learning

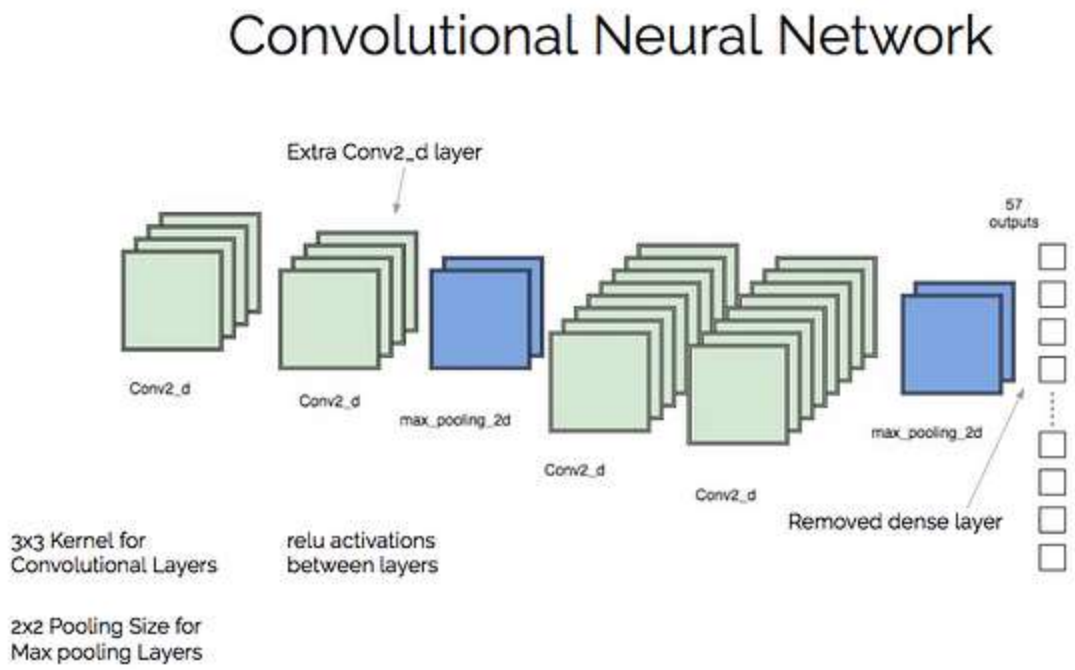
Our Model

We create a CNN by modifying an existing Cifar-10 architecture and train it on spectrograms from 57 unique speakers. Using this trained neural network, we extract features by removing the last fully connected layer and feeding outputs of the flatten layer into an SVM in a process known as transfer learning. There was no publicly available pre-trained model for voice classification so we create and train our own neural network.



CNN Architecture

The green layers in our architecture are convolutional layers whereas the blue layers max pooling. For all convolutional layers we use a 3x3 kernel. For the max pooling we use a pool size of 2x2. We use relu activation functions between each layer and a softmax activation function for the last layer. Our loss function is categorical cross-entropy.



Modified Cifar-10 Architecture

CNN Training

When trained on 6 different people, the neural network is 97% accurate. Each of the 6 people had about an hour of audio that the CNN was trained on. After our data scraping and processing is done for a larger dataset of 162 different speakers, we trained our neural network on 57 different speakers due to time constraints on training and storage space on AWS. We trained each of the 57 speakers on 45 min worth of audio (~2700 seconds). After one epoch our CNN is 97% accurate. The CNN took about an and hour and a half to train on ~24000 spectrogram images.

```
from keras.callbacks import ModelCheckpoint
filepath="/home/ubuntu/weights.best.hdf5"
ep = 0
while True:
    print("Epoch number : " + str(ep))
    model.fit(np.array(x_new_train), dummy_y,
              batch_size=30,
              epochs=1,
              verbose=1,
             validation_data=(np.array(x_test[:1000]), dummy2_y[:1000]))
    model.save_weights('my_model_weights.h5')
```

```
Epoch number : 0
Train on 24444 samples, validate on 1000 samples
Epoch 1/1
24444/24444 [==============================] - 5026s - loss: 1.2946 - acc: 0.6696 - val_loss: 0.3154 - val_acc: 0.901
0
Epoch number : 0
Train on 24444 samples, validate on 1000 samples
Epoch 1/1
  690/24444 [..............................] - ETA: 4842s - loss: 0.0989 - acc: 0.9725
```

SVM and Transfer Learning

We now have have a decent neural network at identifying 57 different people. We cut off the last layer which is a dense layer classifying 57 people, and use the flatten layer to feed into an SVM. SVMs are supposed to perform well with smaller amounts of data (compared to a neural network) and with high dimensions. Using our CNN as a feature extractor we have data in ~400,000 dimensions. We use a radial basis function as the kernel for the SVM.

Using 35 seconds of audio for training on 3 different speakers and testing on 35 seconds results in 95% accuracy. Feeding into the SVM we see that with 15 seconds training for each of 3 different speakers and testing on 15 seconds for each speaker, our SVM results in an accuracy of 83%. We see that we are able to learn someone's voice in 15–20 seconds now as opposed to 45 minutes of audio.

More Examples and Results

All our examples will try to differentiate between three new voices.

When we first tested the SVM we tested it on three YouTubers with 7 samples for each the training and test set. We had an accuracy of 95%.

The indices correspond to a specific person. The array in this example is set up such;

[Christen Dominique, Tushar, Sriraj Raval]

An output of 0 is Christen (Female) , 1 is Tushar (Male) and 2 is Sriraj (Male).

```
In [74]: print(svm_y_test)
         [0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1]

In [75]: print(clf.predict(svm_x_test))
         [0 0 0 0 0 0 0 2 2 2 2 2 2 2 1 1 2 1 1 1 1]

In [77]: from sklearn.metrics import accuracy_score
         accuracy_score(svm_y_test, clf.predict(svm_x_test))

Out[77]: 0.95238095238095233
```

We see here that the model never misclassifies Christen, but classified Sriraj as Tushar for one sample. For future testing we tried to minimize number of samples for training to about 5 samples. The number of times the a person name occurs in the array is how many samples there were for the test set.

For testing our program we made an interface where we record speakers and create a test and train set. We use this program to run live demos and test on real people, not just scraped data.

Additionally our classifier is language agnostic; it can recognize your voice independent of language. The order a name appears in the array is the index the classifier predicts as the person speaking. The first array seen is the prediction and the second array is the true speaker.

Below are the results of a few live tests of our model.

When doing the live demo in our class, learning three people's voices, two males and one female, we reached 63% accuracy. This was based on 5 samples for training and 3 samples for testing. Below is an example of the in class demo we did. (0 → caramanis, 1 → dimakis, 2 → monica)

```
[0 0 2 0 1 0 0 2 2 2 2]
['carmanis', 'carmanis', 'carmanis', 'dimakis', 'dimakis', 'dimakis', 'dimakis', 'monica', 'monica'
, 'monica', 'monica']
0.636363636364
```

In class live demo with our professors and teaching assistant with some mixed language

Another example consisted of two males and one female voice, where all switched between english and a respective different foreign language (Spanish, Arabic and Urdu). Our model was 90% accurate.

```
[0 1 0 1 1 1 1 2 2 2 2]
['hamza', 'hamza', 'hamza', 'isabel', 'isabel', 'isabel', 'isabel', 'yousef', 'yousef', 'yousef', '
yousef']
0.909090909091
```

Demo among friends mixing english and their native languages

Here is another example with 86% accuracy.

```
model trained
[0 0 1 0 2 1 1 1 1 1 2 2 2 2 2]
['chris', 'chris', 'chris', 'chris', 'chris', 'isabel', 'isabel', 'isabel', 'isabel', 'isabel', 'yo
usef', 'yousef', 'yousef', 'yousef', 'yousef']
0.866666666667
```

Demo among friends in purely english

Results were generally positive! Testing on groups of 3 with differing genders generally yields accuracy of 60–90%. However, our model does have limitations. Performance declines when tested on groups of entirely the same gender or as group size increases. Testing of groups of uniform genders generally yields accuracy of 40–60%. Accuracy nears that of a random guess as group size surpasses 6.

V. Conclusion

Summary

We wanted to create a model to identify speakers with only a few sentences of training data. We chose to approach this by using existing image classifying architectures, representing audio using spectrograms. This included a significant data collection component, leading us to create a dataset of 162 speakers included segmented audio files and segmented spectrograms. We chose to use take a transfer learning approach, training a derivative of the Cifar-10 CNN, and extracting features to feed an SVM to classify new speakers. We restricted our training data to 20–35 seconds per person (4–7 samples). This method brought surprising levels of accuracy (60–90%) for groups of 3 with differing genders. Results were less impressive for uniformly gendered groups, but consistently much better than random guesses.

Contributions

Following the goals we set upon starting this project, our team successfully managed to make original contributions to this area of research. We managed to create an extremely large set of audio download links for unique speakers in a field where lack of open source data is a common hurdle to research projects. Again, this list of links can be found here. Additionally, our approach of using image recognition in conjunction with transfer learning with an SVM for audio data has not been heavily explored. It is our hope that our architecture and methods may be useful to future research.

Note: Link to full data set including audio and spectrograms coming soon...

Future Changes/Improvements

A lot of roadblocks for this project consisted of time and computer resources such as storage and computing power. Below are possible future steps for our team or someone else wanting to improve what we have worked on

First would be access to more storage so we can train our neural network on 4 hours of audio per person and use all 162 speakers we scraped. We believe that this will make for an even better feature extractor to feed into the SVM.

Second, do some feature selection before feeding the features into the SVM. Even though SVMs with nonlinear kernels are resistant to over-fitting, having so many features with so few samples may have resulted in over-fitting, which may explain the high variance in accuracy scores. With more time, we would have done more experimentation with the architecture of the neural network, specifically adding another dense layer before the classification layer. This would reduce the input to the SVM from ~400,000 features to whatever we set as the number of nodes in the new dense layer. Another architecture to explore would be basing a model on VGG19.

Third, would be to scrape more speakers. We are able to scrape 6000+ unique speakers from LibriVox, although this takes time to download and preprocess data and a ridiculous amount of storage. We would try to scrape about 500–1000 and use about 30–45 min. of audio per person to see how this improves our feature extractor. This would take a lot of time to train. For reference 57 speakers with 45 min. of audio took an hour and a half to train.

https://medium.com/hackernoon/hello-from-the-mobile-side-tensorflow-lite-in-speaker-recognition-7519b18c2646

"Hello," from the Mobile Side: *TensorFlow Lite in* Speaker Recognition

The Alibaba tech team explored a new approach to voice recognition on mobile, addressing main challenges in this field

Voice biometrics, or voiceprints, are already used by banks such as Barclays and HSBC to verify customer identity. As the technology improves, it will likely find further applications within banking and security. Speaker recognition may also find applications within surveillance criminal investigations.

Despite this potential of the technology, there are still many challenges to overcome in the field. Currently most speaker recognition takes place on the server side. The Alibaba tech team proposes a solution using TensorFlow Lite on the client side, to address many of the common issues with the current model through machine learning and other optimization measures.

Issues with a Server-side Model:

With most speaker recognition currently taking place on the server side, the following issues are all too common:

· Poor network connectivity

· Extended latency

· Poor user experience

· Over-extended server resources

Alibaba's Client-side Alternative:

To address these issues, the Alibaba tech team decided to implement speaker recognition on the client side, and to use machine learning to optimize speaker recognition.

This solution came with its own fair share of challenges. Implementing speaker recognition on the client side is time-consuming, and multiple optimizations are needed in order to offset this. The Alibaba tech team devised ways to:

· Optimize results with machine learning

· Accelerate computation

· Reduce time-consuming operations

· Reduce preprocessing time

· Filter out non-essential audio samples

· Remove unnecessary computing operations

The methods proposed by the Alibaba tech team make up a set of solutions which can help to address the challenges encountered by speaker recognition technology.

Defining Speaker Recognition

Scenarios

Voice recognition can be usefully applied in many scenarios, including:

1) **Media quality analysis:** recognize human voices, silence in call, and background noises.

2) **Speaker recognition:** verify a voice for phone voice unlock, remote voice identification, etc.

3) **Mood recognition:** identify the speakers mood and emotional state. When combined with a person's voiceprint, the content of what is being said, mood recognition can add to security and prevent voiceprint counterfeiting and imitation.

4) **Gender recognition**: distinguish whether a speaker is male or female. This is another tool which can be useful to help confirm speaker identity.

Process

Training and prediction are the two main stages of speaker recognition. The training stage sets up a compute model with the old data, and the prediction gives the reference result on current data with the model..

Training can be further divided into three steps:

1) Extract audio features with Mel-frequency cepstrum algorithm.

2) Mark human voices as positive and non-human samples as negative. Train the neural network model with these samples.

3) Use the final training results to create the prediction model for mobile.

In short, the training flow is feature extracting, model training and mobile model porting. For prediction, the flow is extract the voice feature, run the mobile model with the feature, and get the final predict result.

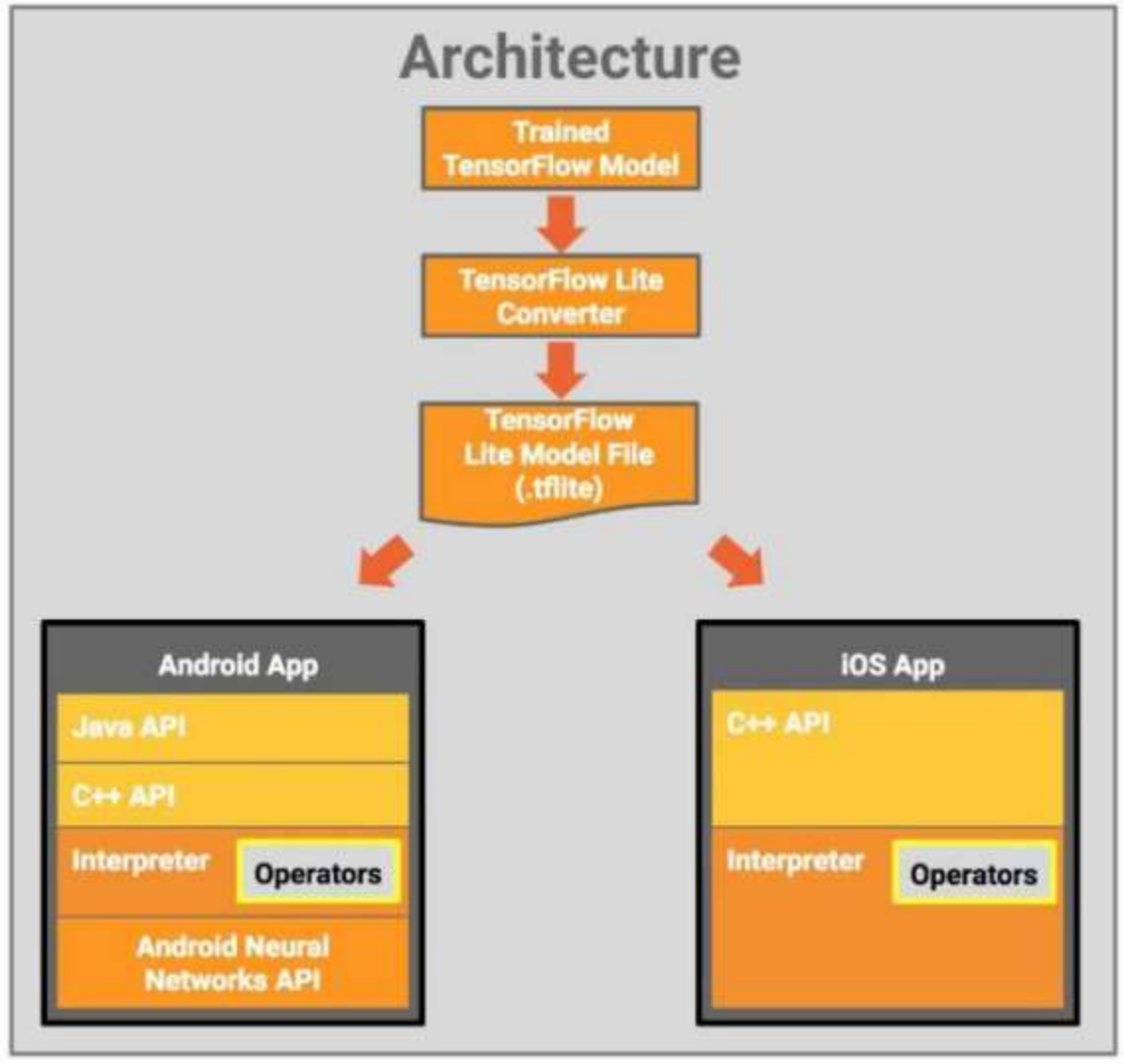
Artificial Intelligence Framework

The launch of TensorFlow Lite was announced at the Google I/O annual developer conference in November 2017. TensorFlow Lite is a lightweight solution for mobile and embedded devices, and supports running on multiple platforms, from rackmount servers to small IoT devices.

With the widespread use of machine learning models, there has been a demand to deploy TensorFlow Lite on mobile and embedded devices. Fortunately, TensorFlow Lite allows machine learning models to run on devices with low latency inference.

Tensorflow Lite is an AI learning system from Google. Its name derives from its operating principles. Tensor means N-dimensional array, flow implies calculation based on data flow graphs. TensorFlow refers to the calculation process of tensor flowing from one end of the data flow graph to the other. TensorFlow is a system that transmits complex data structures to AI neural networks for analysis and processing.

The following figure shows the TensorFlow Lite data structure :



TensorFlow Lite architecture diagram

Mel-Frequency Cepstrum Algorithm

Algorithm introduction

For this solution, the Alibaba tech team uses the Mel-frequency cepstrum algorithm. The algorithm is used for speaker recognition. Using the algorithm contains the following steps:

1) Input sound files and resolve them to original sound data (time domain signal).

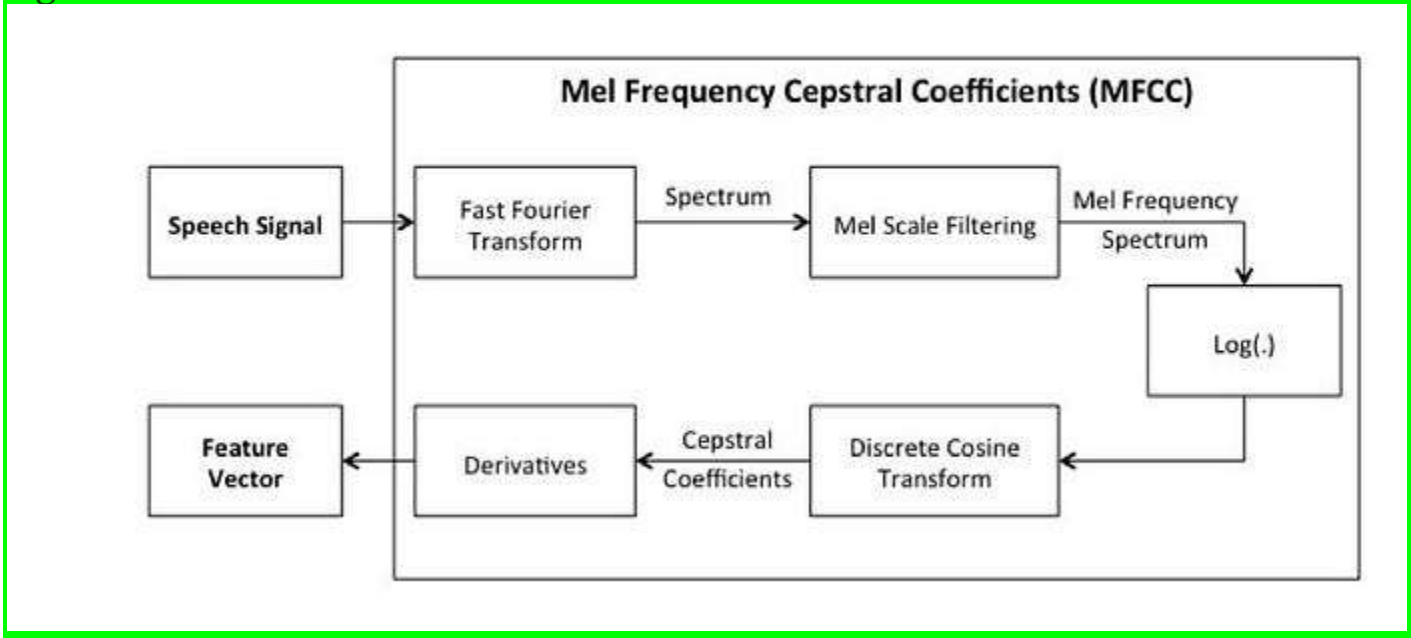
2) Convert time domain signals to frequency domain signals through short-time Fourier transform, windowing and framing.

3) Turn frequency into a linear relationship that humans can perceive through Mel spectrum transform.

4) Separate the DC component from the sine component by adopting DCT Transform through Mel cepstrum analysis.

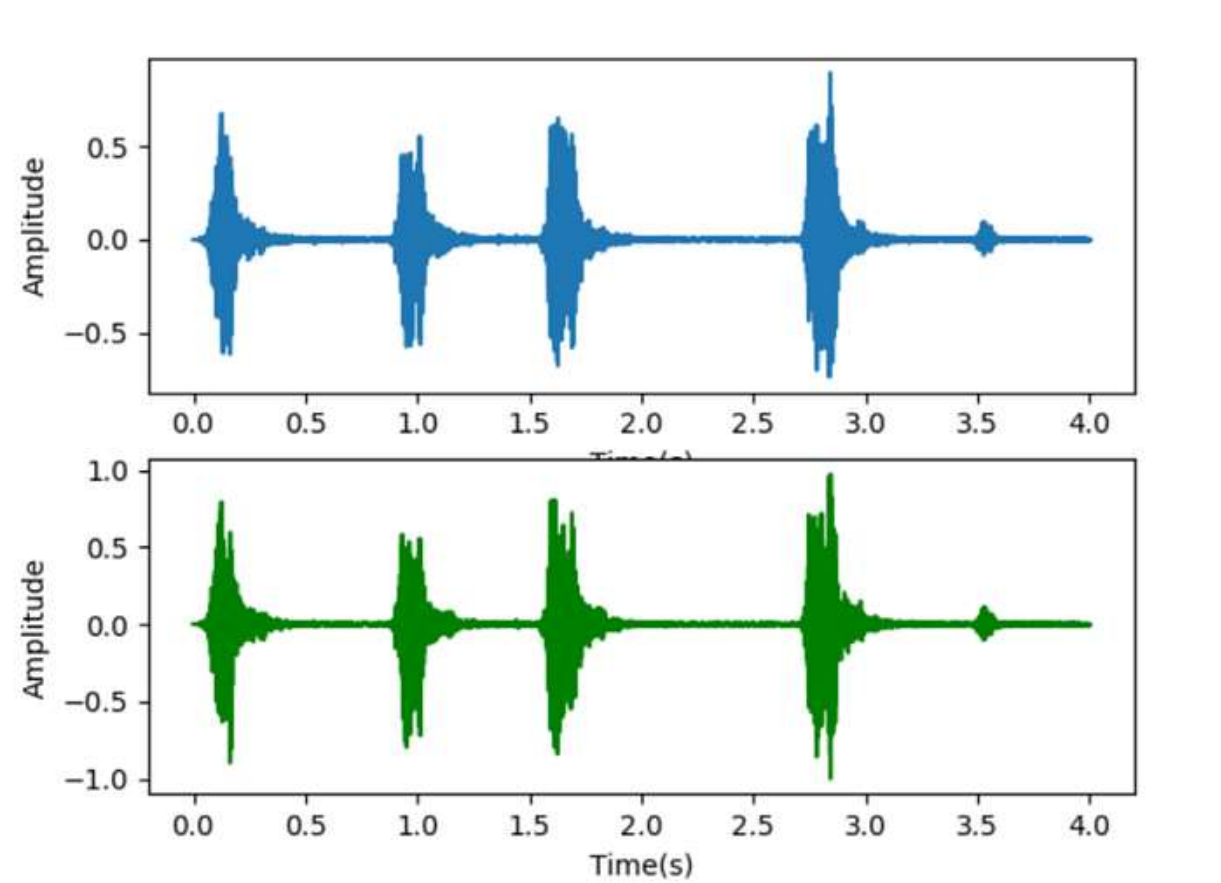
5) Extract sound spectrum feature vectors and convert them to images.

The purpose of windowing and framing is to ensure the short-term stationary character of the speech in the time domain. Mel spectrum transform is used to translate human auditory perception of frequency into a linear relationship. Mel cepstrum analysis is used to understand Fourier transform, through which, any signal can be decomposed into the sum of a DC component and a number of sine signals.

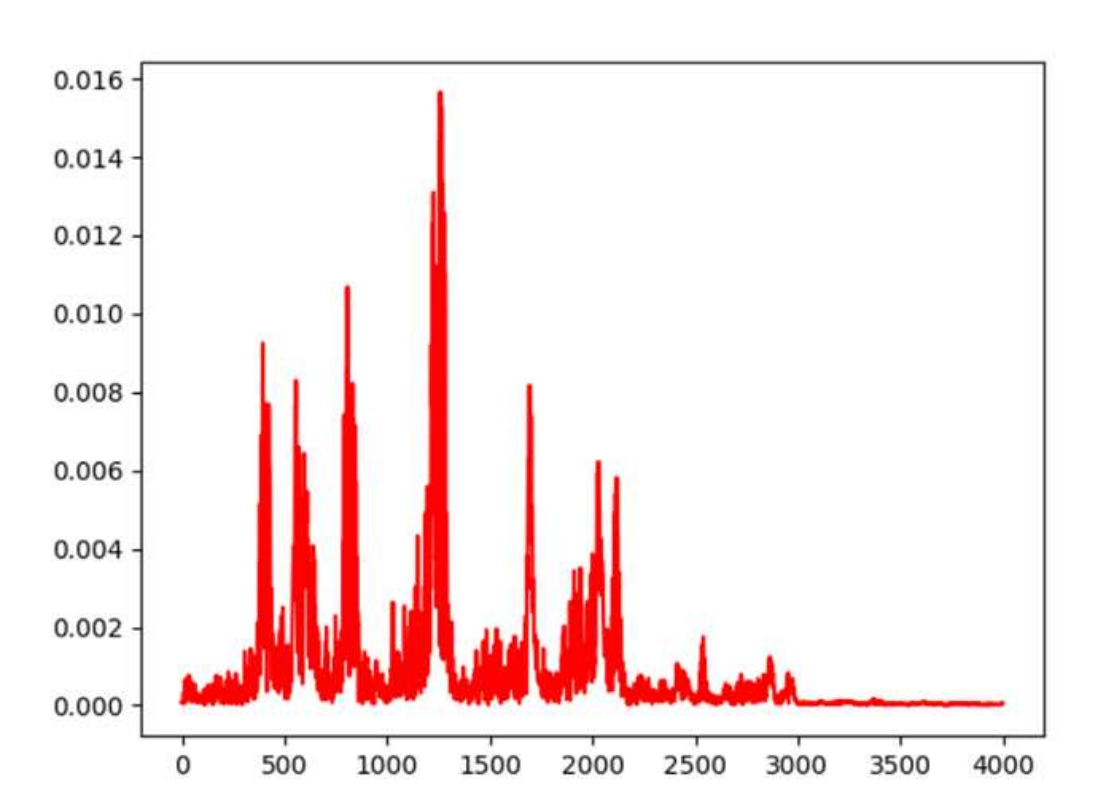


The Mel-frequency cepstrum algorithm implementation process

Short-time Fourier transform

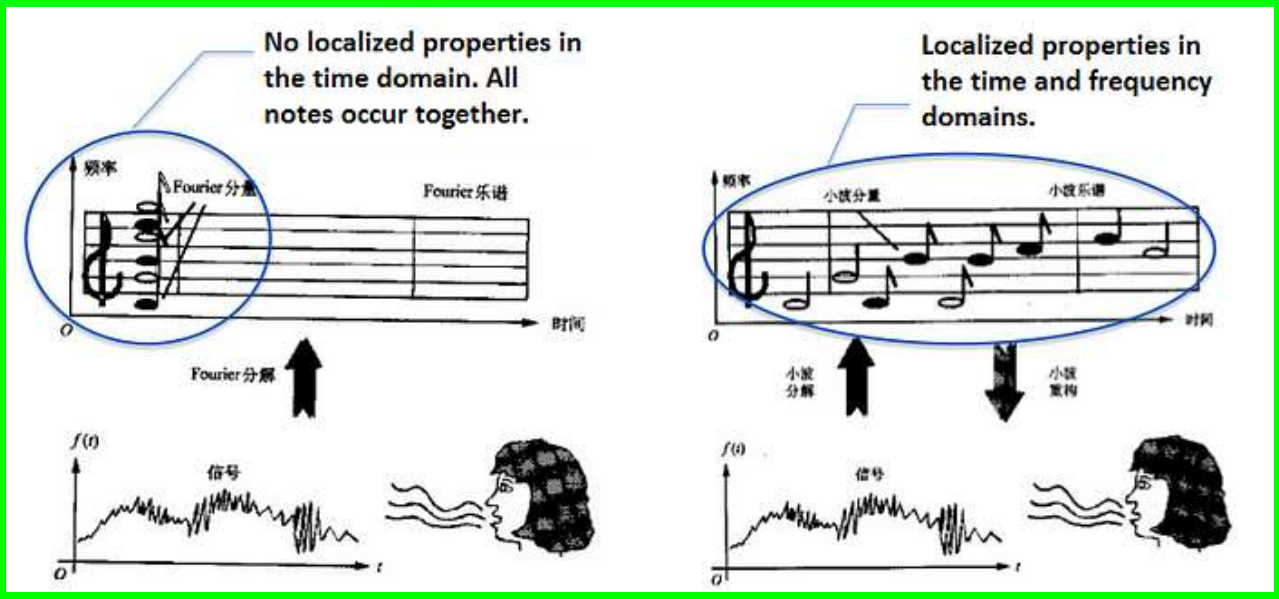


Time domain sound signals



Frequency domain sound signals

The sound signal is a one-dimensional time domain signal. It is difficult to find the rule of how the frequency changes. If we convert the sound signal to frequency domain via Fourier transform, it will show the signal frequency distribution. But at the same time, its time domain information will be missing, making it impossible to see the change of frequency distribution over time. Many joint time-frequency analysis methods have emerged to solve this problem. Short-time Fourier transform, wavelet, and Wigner distribution are all frequently-used methods.



FFT transform and STFT transform.

The signal spectrum can be obtained via Fourier transform and can be widely utilized. For example, signal compression and noise reduction can both be based on the spectrum. However, Fourier transform is built upon an assumption that the signal is stationary, that is, that the statistical properties of the signal do not change over time. However, the sound signal is not stationary. Over a long period of time, there are many signals that will appear and then disappear immediately. If all the signals are Fourier transformed, the change of sound over time cannot be reflected accurately.

The short-time Fourier transform (STFT) used in this article is the classic joint time-frequency analysis method. Short-time Fourier transform (STFT) is a mathematical transformation associated with the Fourier transform (FT) to determine the frequency and phase of a sine wave in a local region of the time-varying signal.

The concept of short-time Fourier transform (STFT) is to first choose a window function with time-frequency localization, then assume that the analysis window function h (t) was stationary over a short time, which ensures f (t) h (t) is a stationary signal within different finite time widths. Finally, calculate the power spectrum at various moments. STFT uses fixed window functions, the most commonly used of which include the Hanning window, the Hamming window, and the Blackman-Haris window. The Hamming window, a generalized cosine window, is used in the solutions in this article. The Hamming window can efficiently reflect the attenuation relationship between energy and time at a certain moment.

The STFT formula in this article takes the original Fourier transform formula,

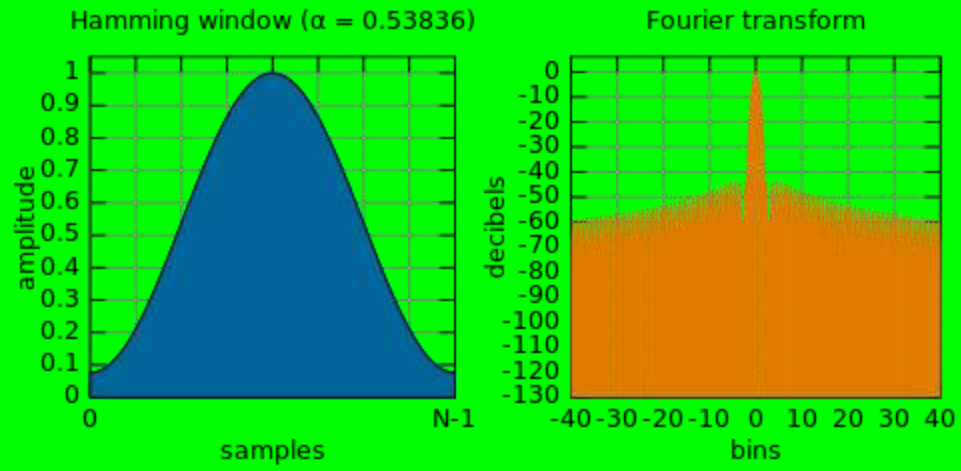
$$\mathrm{F}(\omega)=\int_{-\infty}^{+\infty} f(\tau)e^{-j\omega\tau}d\tau$$

and adds a window function to it, creating the following updated STFT formula:

$$\mathrm{F}(\omega,t)=\int_{-\infty}^{+\infty} f(\tau)h(\tau-t)e^{-j\omega\tau}d\tau$$

The following is a Hamming window function:

$$h(n) = 0.53836 - 0.46164\cos(\frac{2\pi n}{N-1}), 0 \le n \le N-1$$



STFT transform based on the Hamming window

Mel Spectrum

Spectograms are usually in the form of a large map. In order to turn the sound features into a suitable size, they often need to be transformed into Mel spectrum via Mel scale filter bank.

Mel scale

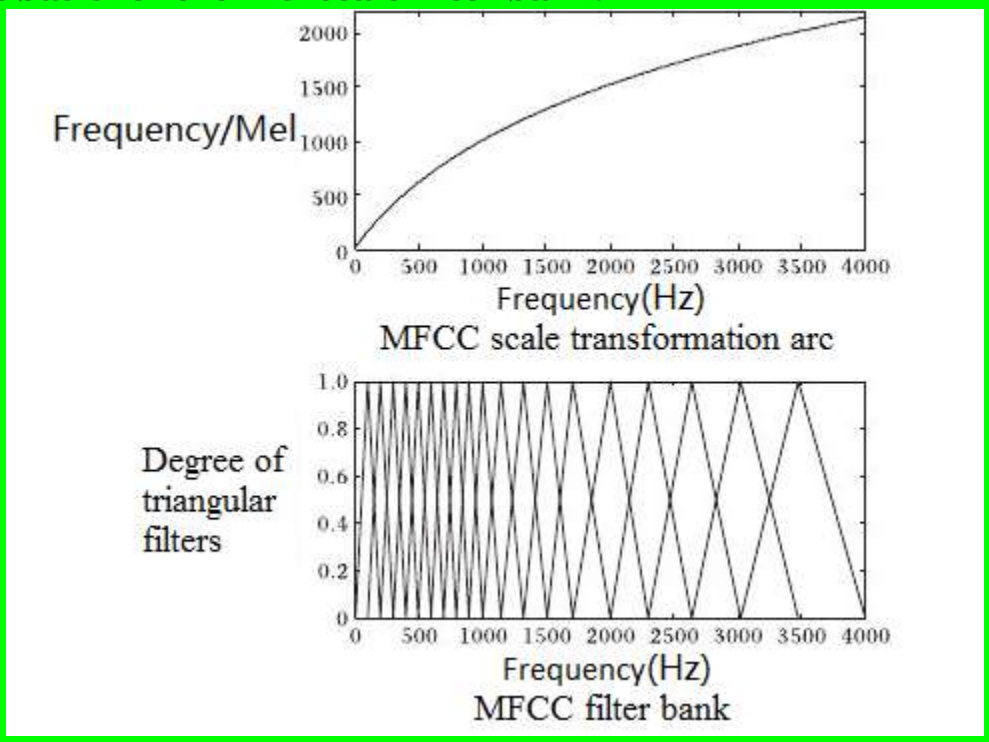
The Mel scale was named by Stevens, Volkmann, and Newman in 1937. It is known that the unit of frequency is Hertz (Hz) and the frequency range of human hearing is 20–20000 Hz.

But human auditory perception does not relate to scale units such as Hz in a linear manner. For example, if we have adapted to a 1000Hz tone, then when tone frequency is increased to 2000Hz, our ears could only perceive that the frequency may be increased by a little, and we would never realize that the frequency had doubled. The mapping for converting an ordinary frequency scale to Mel-frequency scale is as follows:

$$mel(f) = 2595 * \log_{10}(1 + f / 700)$$

The above formula changes the frequency so that it has a linear relationship with human auditory perception. That is to say, if one Mel scale frequency is the double of another Mel scale frequency, human ears could perceive that one frequency is roughly the double of the other.

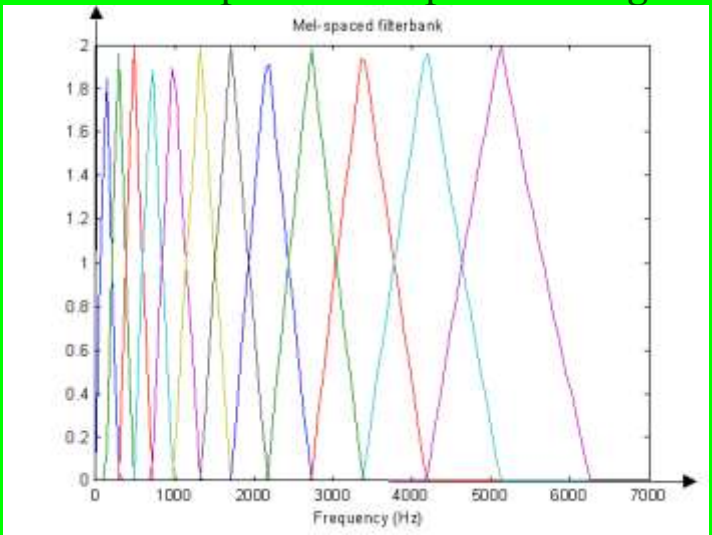
Since there is a log relationship between Hz and Mel frequency, if the frequency is low, Mel-frequency will change rapidly with Hz; if the frequency is high, Mel-frequency will change slowly. This shows that human ears are sensitive to low frequency sounds and less responsive to high frequency sounds. This rule forms an important basis for the Mel scale filter bank.



Frequency transforms to Mel frequency

The figure below shows 12 triangular filters forming a filter group. This filter group features dense filters and high threshold in the low frequency zone, as well

as sparse filters and low threshold in the high frequency zone. This aligns well with the fact that human ears are less responsive to sounds of higher frequency. Mel-filter banks with same bank area, a form of filters shown in the figure above, are widely used in fields such as speech and speaker recognition.



Mel filter bank

Mel-Frequency Cepstrum

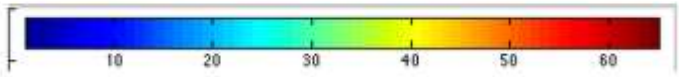
The result of applying DCT transformation to the Mel log spectrum to separate components of DC signal and sine signals is the Mel-frequency cepstrum (MFC).

$$mfcc(u) = c(u)\sum_{i=0}^{N-1} mel(i)\cos[(\frac{(i+0.5)\pi}{N})u]$$

Where, $c(u) = \begin{cases} \sqrt{\frac{1}{n}}, u = 0 \\ \sqrt{\frac{2}{n}}, u \neq 0 \end{cases}$

MFC exports vectors that cannot be displayed with pictures before they are transformed to image matrices. The scope of exported vectors $mel \in [\min, \max]$ must be linearly transformed to the scope of images $pixel \in [0, 255]$

$$pixel = \frac{mel - \min}{\max - \min} \times 255$$



Drawing color scale

Optimizing algorithm processing speed

Since the algorithm is designed to be used on the client side, fast processing is required. The following steps can be taken to optimize algorithm processing speed.

1) **Instruction set acceleration:** The algorithm features many matrix addition and matrix multiplication operations. The ARM instruction set is introduced to accelerate operations. It can increase the speed by 4–8 times.

2) **Algorithm acceleration:**

a) Select vocal frequency range (20HZ~20KHZ) and filter out input outside the non-vocal frequency range to reduce redundant computation.

b) Lower the audio sampling rate to reduce unnecessary data computation. This is possible because human ears are insensitive to sampling rates that are too high.

c) Cut windows and sections reasonably to avoid excessive computation.

d) Detect silent sections so that unnecessary sections can be deleted.

3) **Sampling frequency acceleration:** If radio sampling frequency is too high, choose down sampling and set the highest sampling frequency to 32 kHz.

4) **Multi-thread acceleration:** Divides audios into multiple fragments concurrently processed by multi-threads. The number of threads depends on the machine capacity. The default setting is four threads.

```
// Global variables
const int _FS=32;                       // default down sampling rate in KHz
const int _HIGH=20;                      // default high frequency limit in KHz
const int _LOW=0;                       // default low frequency limit in KHz
const int _FrmLen=25;                  // frame length in ms
const int _FrmSpace=10;                // frame space in ms
const unsigned long FFTLen=2048;       // FFT points
const double PI=3.1415926536;          // number of PI
const int FiltNum=128;                 // number of filters
const int PCEP=128;                    // number of cepstrum
const int num_mfcc_threads = 4;        // number of mfcc thread
const int max_cut_period = 4;             // max number of cut period
```

Algorithm parameters selected by the engineering team

Speaker Recognition Models

Model selection.

Convolutional Neural Networks (CNN) are a type of feedforward neural network. CNN networks contain artificial neurons that can respond to some of the neurons in their field. This type of neural network is highly suitable for processing large images.

In the 1960s, while studying neurons in the cerebral cortices of cats that aid in local sensing and direction selection, Hubel and Wiesel found unique cellular structures that could be used to simplify feedback neural networks. This led them to propose the CNN concept. CNNs have become a hot spot in many research fields, particularly in modal classification.

CNNs owe part of their popularity to their ability to skip complex pre-processing of images and allow direct import of original images. The first CNN-like network was the neocognitron proposed by K.Fukushima in 1980. Since then, multiple researchers have worked to improve the CNN model. Among the most significant of these efforts has been the work on improved cognition proposed by Alexander

and Taylor. This research on improved cognition combines the strengths of various approaches and avoids time-consuming error back propagation.

In general, CNN structure is made up of two fundamental layers. One of these layers is the feature extraction layer. In this layer, each neuron's input is connected to the local accepted domain of the previous layer and extracts the features of this local domain. Once extracted, the positional relation of this local domain feature to other features is fixed.

The other is the feature mapping layers which gather to form every computing layer of the network. Each feature mapping layer is a surface in which all neurons have the same weight. Using functions with small influence function kernels, such as sigmoid and relu as the activation function of CNN, the feature mapping structure ensures constant displacement of feature mapping. The number of free network parameters is reduced, since neurons on the same mapping surface share the same weight. In CNN, every convolutional layer is closely followed by a computing layer that is used to obtain local averages and secondary extraction. This specific secondary feature extraction reduces feature resolution.

CNN is used mainly to recognize 2D images that are modified but not deformed, such as images which have been zoomed in on. Since CNN's feature detection layer learns implicitly through training data, using CNN can avoid the need for explicit feature extraction.

CNN can also learn concurrently, as it places the same weight for neurons across the same feature mapping surface. This gives CNN another big advantage over networks in which neurons are inter-connected. Local weight sharing gives CNN a unique advantage in speech recognition and image processing. In terms of layout, CNN is closer to actual biological neural networks. Weight sharing ensures a less

complex network and CNN does not have to handle the complexity of data reconstruction during feature extraction and classification, since images of multi-dimensional input vectors can be directly imported into the network.

Inception-v3 model

The accurate Inception-v3 model is used in this article as the speaker recognition model. Decomposition is one of the most important improvements of the v3 model. 7x7 CNNs are decomposed into 2 one-dimensional convolutions (1x7, 7x1), and 3x3 CNNs are also decomposed into two convolutions (1x3,3x1). This accelerates computation, further deepens the network, and makes it more non-linear. The v3 model network input is upgraded from 224x224 to 299x299, and the design of the modules 35x35/17x17/8x8 are improved.

Using a TensorFlow session enables modules to realize training and prediction at the code layer. The TensorFlow official website provides details of how to use a TensorFlow session.

```
# Using the `close()` method.
sess = tf.Session()
sess.run(...)
sess.close()

# Using the context manager.
with tf.Session() as sess:
  sess.run(...)
```

Using a TensorFlow session

Model example.

In monitored machine learning, samples are typically divided into three sets:

· **The train set:** This set is used to estimate models. It learns the sample data sets and builds a classifier by matching some parameters. It creates a classification manner used to train models.

· **The validation set:** This set is used to determine the network structure or control parameters that control model complexity. It adjusts the classifier parameters of models obtained through learning, such as choosing to hide the number of units in neural networks. The validation set is also used to determine the network structure or parameters that control the complexity of models, in an attempt to avoid overfitting of models.

· **The test set:** This set is used to check how the finally selected optimal model performs. It is mainly used to test the recognition capacity (such as the recognition rate) of trained models.

The Mel Frequency Cepstrum algorithm described in the second chapter can be used to obtain speaker recognition as sample files. Sounds within the human vocal spectrum as positive samples, animal sounds and other non-human noises are used as negative samples. These samples are then used to train the Inception-v3 model.

With TensorFlow as the training frame, this article takes 5000 human vocal samples and 5000 non-human vocal samples as the test sets, and 1000 samples as the validation set.

Model training.

Once samples are ready, they can be used to train the Inception-v3 model. The convergence of the trained model can generate the pb model usable on the client. In model selection, choose compiling armeabi-v7a or a later version, and NEON optimization is enabled by default. In other words, opening the macro of USE_NEON can accelerate instruction sets. More than half of the operations in a CNN take place at convolutions, so instruction set optimization can speed up operations by at least four times.

```
TfLiteRegistration* Register_CONV_2D() {
#ifdef USE_NEON
  return Register_CONVOLUTION_NEON_OPT();
#else
  return Register_CONVOLUTION_GENERIC_OPT();
#endif
}
```

Convolution processing function

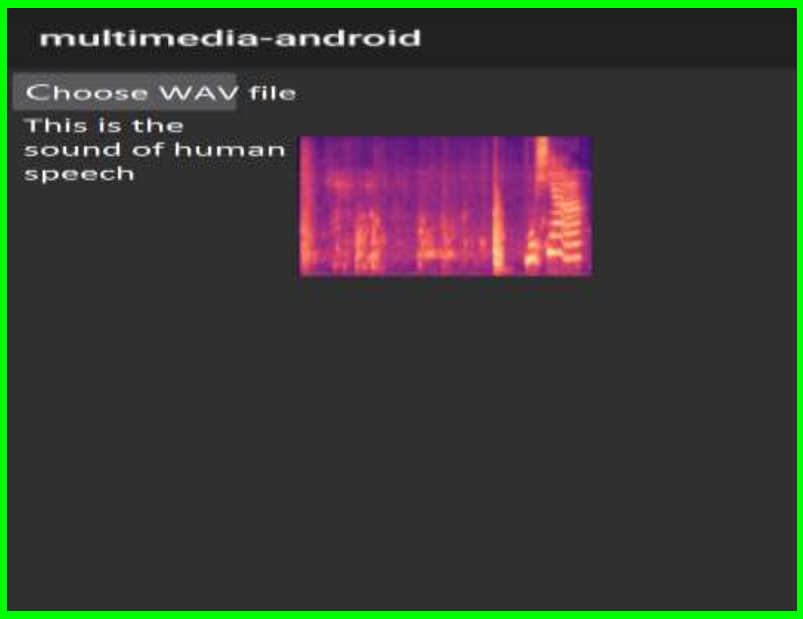
The toco tool provided by TensorFlow can then be used to generate a lite model which can be directly called by the TensorFlow Lite frame on the client.

```
tf.contrib.lite.toco_convert(
    input_data,
    input_tensors,
    output_tensors,
    inference_type=FLOAT,
    input_format=TENSORFLOW_GRAPHDEF,
    output_format=TFLITE,
    quantized_input_stats=None,
    drop_control_dependency=True
)
```

Toco calling interface.

Model prediction.

MFC can be used to extract vocal sound file features and generate prediction images. Using the lite prediction model generated in training produces the following results:



Model prediction result

Conclusion

The methods proposed above can help to address some of the most difficult challenges in the speaker recognition field today. Using TensorFlow Lite on the client side is a useful innovation that helps leverage machine learning and learning and neural networks to drive the technology forward to further progress. (Original article by Chen Yongxin陈永新)

References:

[1] https://www.tensorflow.org/mobile/tflite

[2] 基于MFCC与IMFCC的说话人识别研究[D]. 刘丽岩. 哈尔滨工程大学 . 2008

[3] 一种基于MFCC和LPCC的文本相关说话人识别方法[J]. 于明,袁玉倩,董浩,王哲. 计算机应用. 2006(04)

[4] Text dependent Speaker Identification in Noisy Enviroment[C]. Kumar Pawan,Jakhanwal Nitika,

Chandra Mahesh. International Conference on Devices and Communications . 2011

[5] https://github.com/weedwind/MFCC

[6] https://baike.baidu.com/item/ARM指令集/907786?fr=aladdin

[7] https://www.tensorflow.org/api_docs/python/tf/Session

Track who's speaking with Speaker Diarization aka Speaker-Recognition

https://medium.com/vmacwrites/track-whos-speaking-with-speaker-diarization-2e3eac2de2c3

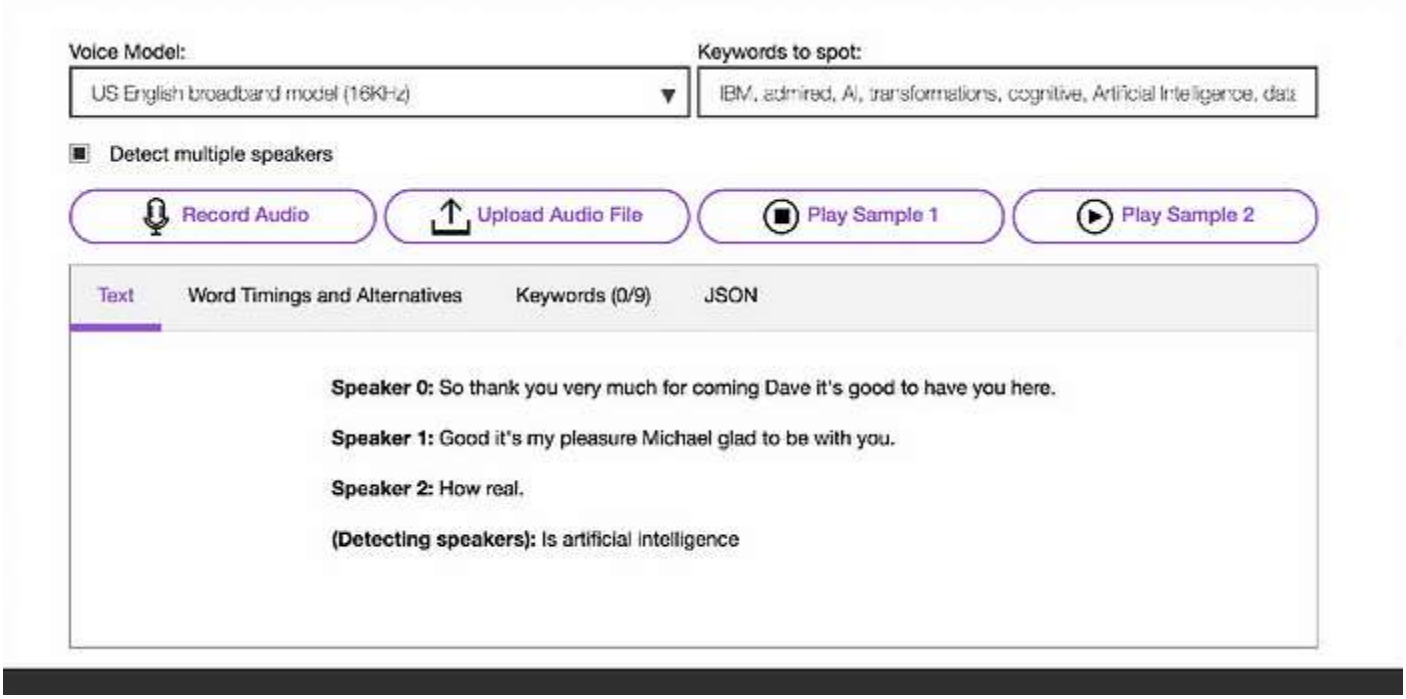
Distinguishing between two speakers in a conversation is difficult especially when you are hearing them virtually or for the first-time. Same can be the case when multiple voices interact with AI/Cognitive systems, virtual assistants, and home assistants like Alexa or Google Home. To overcome this, Watson's Speech To Text API has been enhanced to support real-time speaker 'diarization.'

IBM Watson

Post building a popular chatbot using Watson services called "WatBot", there are a couple of requests to include SpeakerLabels setting into our code sample.

So, what is Speaker Diarization?

Speaker diarization (or diarization) is the process of partitioning an input audio stream into homogeneous segments according to the speaker identity. It can enhance the readability of an automatic speech transcription by structuring the audio stream into speaker turns and, when used together with speaker recognition systems, by providing the speaker's true identity.



Real-time Speaker Diarization with Watson Speech-to-Text API

Why Speaker Diarization?

Real-time speaker diarization is a need we've heard about from many businesses across the world that rely on transcribing volumes of voice conversations collected every day. Imagine you operate a call center and regularly take action as customer and agent conversations happen — issues can come up like providing product-related help, alerting a supervisor about negative feedback, or flagging calls based on customer promotional activities. Prior to today, calls were typically transcribed and analyzed after they ended. Now, Watson's speaker diarization capability enables access to that data immediately.

To experience speaker diarization via Watson speech-to-text API on IBM Bluemix, head to this demo and click to play sample audio 1 or 2. If you check the input JSON below; we are setting "speaker_labels" optional parameter to true. This helps us in distinguishing between speakers in a conversation.

```
{
 "continuous": true,
 "timestamps": true,
 "content - type": "audio / wav",
 "interim_results": true,
 "keywords": ["IBM", "admired", "AI", "transformations",          "cognitive",
```

```
"Artificial Intelligence", "data", "predict", "learn"],

 "keywords_threshold": 0.01,
 "word_alternatives_threshold": 0.01,
 "smart_formatting": true,
 "speaker_labels": true,
 "action": "start"
}
```

A part of output JSON after real-time speech-to-text conversion:

```
{
....
 “confidence”: 0.927,
 “transcript”: “So thank you very much for coming Dave it’s good to have you here. “
}],
“final”: true,
“speaker”: 0
}
```

You can see that a speaker label is getting assigned to each speaker in the conversation.

Steps to enable speaker diarization

- Watson speech-to-text is available as a service on Bluemix, IBM Cloud platform. Create now to leverage the service in your application.
- If you are taking the Rest API approach, don’t forget to include the optional parameter “speaker_labels: true” in your request JSON.
- Based on the programming language your application is created, use any of the SDKs available on Watson Developer Cloud ranging from Python, Node, Java, Swift etc.,

Refer WatBot repository to get a gist of how to enable or add speaker diarization to an existing android app. Similarly, you can use other SDKs to achieve speaker diarization.

Note: Speaker labels are not enabled by default. Check ToDos in the code to uncomment.

Use cases

From integrating into chatbots to interacting with home assistants like Alexa, Google Home etc., From call centers to medical services. The possibilities are endless.

Speaker Recognition using Deep Learning

https://medium.com/@yaduvanshiharsh15/speaker-recognition-using-deep-learning-890fe812a976

Photo by Markus Spiske on Unsplash

Introduction

In today's digital era, voice-based interactions have become increasingly prevalent. From voice assistants like Siri and Alexa to authentication systems and security applications, accurately identifying and recognizing speakers plays a pivotal role in enhancing user experiences and ensuring secure access. This is where speaker recognition, a fascinating field at the intersection of artificial intelligence and signal processing, comes into play.

For starters, speaker recognition can be understood as a technique to recognize who is speaking. It is sometimes also known as voice recognition, voiceprint recognition, or talker recognition. However, it's important **not to confuse speaker recognition with speech recognition.** While both involve the analysis of speech signals, the former is concerned with identifying the individual behind the voice, whereas the latter focuses on transcribing and understanding the content of the spoken words.

Traditionally, speaker recognition relied on statistical modeling techniques such as Gaussian Mixture Models (GMM) and Hidden Markov Models (HMM). While these methods have served their purpose, recent advancements in deep learning have revolutionized the field, enabling more accurate and robust speaker recognition systems.

In this article, we delve into the world of speaker recognition using deep learning. We explore the underlying principles, methodologies, and techniques that empower these systems to decipher the unique characteristics of individual voices. From data preparation and model architectures to training strategies and evaluation metrics, we aim to provide a comprehensive overview of the key components involved in developing effective speaker recognition systems.

Data Preparation

One of the key ingredients for developing a successful speaker recognition model is a high-quality dataset. In our case, we utilized the widely acclaimed LibriSpeech Automatic Speech Recognition (ASR) corpus. This dataset offers a rich collection of audio recordings encompassing a diverse range of speakers and speech content.

The LibriSpeech dataset comprises approximately 1500 books, recorded by 251 different speakers. Each audio file in the dataset is in the FLAC format, ensuring lossless compression and maintaining the fidelity of the speech signals. To maintain consistency and facilitate organization, the dataset follows a specific naming convention for each audio file. For example, consider the file named `103-1240-0000.flac`. Here, the speaker ID is "103," the book ID is "1240," and the utterance ID is "0000." An utterance represents an audio clip of a speech signal, typically corresponding to a sentence or a phrase spoken by the speaker.

To begin building our model, the first step is to create a dictionary that associates each speaker with their corresponding utterances within the training dataset. We will refer to this dictionary as `speaker_to_utterance`. By organizing the data in this manner, we establish a clear mapping between speakers and their audio files, enabling efficient data retrieval during training and evaluation.

Let's take a look at a code snippet that demonstrates this data preparation step:

```
def get_librispeech_speaker_to_utterance(data_dir):
    speaker_to_utterance = dict()
    flac_file = glob.glob(os.path.join(data_dir, "*", "*", "*.flac"))

    for file in flac_file:
        speaker_id = file.split("\\")[-3]
        utterance_id = file.split("\\")[-1].split(".")[0]
        if speaker_id not in speaker_to_utterance:
            speaker_to_utterance[speaker_id] = []
        speaker_to_utterance[speaker_id].append(file)
    return speaker_to_utterance
```

In this code snippet, we traverse through the directory structure of the training dataset, iterating over each speaker, chapter, and audio file. We extract the necessary information, such as the speaker ID and utterance ID, and store the corresponding file path in the `speaker_to_utterance` dictionary. At the end of the process, we obtain a comprehensive dictionary containing all the speakers in the dataset as keys, and their respective audio file locations as values. In our case, this dictionary will consist of a total of 251 unique speakers.

Data Processing: Feature Extraction

In speaker recognition, one of the commonly used techniques is **the Triplet loss**, which we will also employ in our model. With Triplet loss, our input consists of three segments: an anchor, a positive, and a negative. The anchor and positive segments belong to the same speaker, while the negative segment belongs to a different speaker. We will delve into the details of triplet loss later in the article.

Now that we understand the structure of our input, let's focus on converting the audio files in FLAC format into features that our model can process effectively. For this purpose, we will utilize **Mel-Frequency Cepstral Coefficients (MFCC)**. While there are other feature extraction techniques available, such as Perceptual Linear Prediction (PLP), Perceptual Non-Linear Cepstral Coefficients (PNCC), and Linear Frequency-Based Energies (LFBE), we will opt for MFCC in our case due to its robustness in handling noise through logarithmic compression.

```
def get_triplet(spk_to_utts):
    """Get a triplet of anchor/pos/neg samples."""
    pos_spk, neg_spk = random.sample(list(spk_to_utts.keys()), 2)
    while len(spk_to_utts[pos_spk]) < 2:
        pos_spk, neg_spk = random.sample(list(spk_to_utts.keys()), 2)
    anchor_utt, pos_utt = random.sample(spk_to_utts[pos_spk], 2)
    neg_utt = random.sample(spk_to_utts[neg_spk], 1)[0]
    return (anchor_utt, pos_utt, neg_utt)
```

The `get_triplet` function randomly selects two different speakers, one for the positive segment and another for the negative segment, from the provided `spk_to_utts` dictionary. It ensures that the

positive speaker has at least two utterances available. Then, it randomly selects one utterance each from the positive speaker for the anchor and positive segments. Additionally, it randomly selects one utterance from the negative speaker for the negative segment. The function returns a triplet containing the paths of the anchor, positive, and negative utterances.

```
def extract_features(audio_file):
    """Extract MFCC features from an audio file, shape=(TIME, MFCC)."""
    waveform, sample_rate = soundfile.read(audio_file)

    if len(waveform.shape) == 2:
        waveform = librosa.to_mono(waveform.transpose())

    if sample_rate != 16000:
        waveform = librosa.resample(waveform, sample_rate, 16000)

    # Mel-frequency cepstral coefficients (MFCCs) are robust to noise bcoz of logarithmic
compression
    features = librosa.feature.mfcc(y=waveform, sr=sample_rate, n_mfcc=myconfig.N_MFCC)
    # the shape of features will be 40 X 441, where 40 represent featues where as 441
represent frames

    return features. Transpose()
```

The `extract_features` function takes an audio file as input and extracts Mel-frequency cepstral coefficients (MFCCs) as features. It first reads the waveform and sample rate from the audio file using the `soundfile.read` function. If the waveform has two channels, it converts it to mono by taking the average of the channels. If the sample rate is not 16000 Hz, it resamples the waveform to have a sample rate of 16000 Hz.

The MFCC features are then computed using the `librosa.feature.mfcc` function, which takes the waveform, sample rate, and the number of desired MFCC coefficients as input. The resulting features are in the shape of a matrix, where each row represents a feature, and each column represents a frame. The matrix is transposed before being returned.

Model Architecture

In order to process continuous audio signals effectively, an LSTM (Long Short-Term Memory) architecture is chosen for the speaker recognition model. LSTMs are well-suited for capturing long-term dependencies in sequential data such as audio. In this architecture, a bidirectional LSTM with three stacked LSTM layers is utilized to enhance the model's performance in capturing temporal patterns.

The model implementation is shown below using the PyTorch framework

```
class LstmSpeakerEncoder(BaseSpeakerEncoder):
    def __init__(self, saved_model=""):
        super(LstmSpeakerEncoder, self).__init__()
        self.lstm = nn.LSTM(
            input_size=myconfig.N_MFCC,      # Number of MFCC coefficients (40)
            hidden_size=myconfig.LSTM_HIDDEN_SIZE,     # Number of hidden units in each
LSTM layer (64)
            num_layers=myconfig.LSTM_NUM_LAYERS,     # Number of stacked LSTM layers (3)
            batch_first=True,
            bidirectional=myconfig.BI_LSTM     # Whether to use a bidirectional LSTM
(True/False)
        )
        if saved_model:
            self._load_from(saved_model)

    def _aggregate_frames(self, batch_output):
        if myconfig.FRAME_AGGREGATION_MEAN:
            return torch.mean(batch_output, dim=1, keepdim=False)
        else:
            return batch_output[:, -1, :]

    def forward(self, x):
        D = 2 if myconfig.BI_LSTM else 1
        h0 = torch.zeros(D * myconfig.LSTM_NUM_LAYERS, x.shape[0],
myconfig.LSTM_HIDDEN_SIZE).to(myconfig.DEVICE)
        c0 = torch.zeros(D * myconfig.LSTM_NUM_LAYERS, x.shape[0],
myconfig.LSTM_HIDDEN_SIZE).to(myconfig.DEVICE)
        y, (hn, cn) = self.lstm(x, (h0, c0))
        return self._aggregate_frames(y)
```

The core component of the architecture is the `self.lstm` layer, which is an LSTM module from PyTorch's `nn` module. It is configured with an `input_size` equal to the number of MFCC coefficients (`myconfig.N_MFCC`), a `hidden_size` of the LSTM units (`myconfig.LSTM_HIDDEN_SIZE`), a number of `num_layers` specifying the stacked LSTM

layers (`myconfig.LSTM_NUM_LAYERS`), and whether it is bidirectional (`myconfig.BI_LSTM`).

During the forward pass, the input `x` is passed through the LSTM layer. The initial hidden state `h0` and cell state `c0` are initialized as tensors of zeros with appropriate dimensions. The output `y` and final hidden and cell states `hn` and `cn` are obtained from the LSTM layer. The `_aggregate_frames` function is then used to aggregate the output frames of the LSTM layer into a fixed-length representation, depending on the value of `myconfig.FRAME_AGGREGATION_MEAN`. The aggregated output represents the speaker embedding for the input audio sequence.

Overall, this model architecture employs a bidirectional LSTM with three stacked LSTM layers for effective speaker recognition from continuous audio signals.

Model Training

```
def train_network(speaker_to_utterance, num_steps, saved_model="", pool=None):
    losses = []
    start_time = time.time()
    encoder = get_speaker_encoder_LSTM()

    #Train
    optimizer = torch.optim.Adam(encoder.parameters(), lr=myconfig.LEARNING_RATE)
    print("Start training")
    for step in range(num_steps):
        optimizer.zero_grad()

        #build batch input
        batch_input = feature_extraction.get_batched_triplet_input(speaker_to_utterance,
myconfig.BATCH_SIZE, pool)
        batch_output = encoder(batch_input)      #batch_output.shape=[24,64*2]
        loss = get_triplet_loss_from_batch_output(batch_output, myconfig.BATCH_SIZE)
        loss.backward()
        optimizer.step()
        losses.append(loss.item())
        print(f"step: {step}/{num_steps} loss: {loss.item()}")

    #     saving model
        if saved_model is not None and (step + 1) % myconfig.SAVE_MODEL_FREQUENCY == 0:
            checkpoint = saved_model
            if checkpoint.endswith(".pt"):
                checkpoint = checkpoint[:-3]
            checkpoint += ".ckpt-" + str(step + 1) + ".pt"
            save_model(checkpoint,encoder, losses, start_time)
```

```
    training_time = time.time() - start_time
    print("Finished training in", training_time, "seconds")
    if saved_model is not None:
        save_model(saved_model, encoder, losses, start_time)
    return losses
```

To train the LSTM-based speaker recognition model, the following steps are performed:

1. Initializing the encoder: The speaker encoder is initialized using the `get_speaker_encoder_LSTM()` function, which retrieves the LSTM-based speaker encoder defined in the previous section.

2. Defining the optimizer: The Adam optimizer is used for training the model. It is instantiated with the parameters of the encoder, and the learning rate is set to `myconfig.LEARNING_RATE`.

3. Training loop: The training loop iterates over `num_steps`, which represents the total number of training steps. In each iteration, the optimizer's gradient is zeroed (`optimizer.zero_grad()`) to clear any accumulated gradients from the previous iteration.

4. Batch input preparation:
   The `feature_extraction.get_batched_triplet_input()` function is used to construct a batch of triplet inputs from the `speaker_to_utterance` dictionary. This function selects random anchor, positive, and negative utterances belonging to different speakers and forms a batch input. The batch size is determined by `myconfig.BATCH_SIZE`.

5. Forward pass and loss calculation: The batch input is passed through the encoder (`encoder(batch_input)`) to obtain the batch output, which represents the speaker embeddings. The triplet loss is computed using

the `get_triplet_loss_from_batch_output()` function, which calculates the loss based on the batch output and the batch size (`myconfig.BATCH_SIZE`).

6. Backpropagation and parameter update: The loss is backpropagated through the model (`loss.backward()`) to compute the gradients of the model parameters. The optimizer then performs a parameter update (`optimizer.step()`) based on these gradients.

7. Loss tracking: The current loss value (`loss.item()`) is appended to the `losses` list to track the training progress.

8. Saving the model: If a `saved_model` path is provided and the current step is a multiple of `myconfig.SAVE_MODEL_FREQUENCY`, the model is saved to a checkpoint file using the `save_model()` function. The checkpoint file name includes the step number for easy identification.

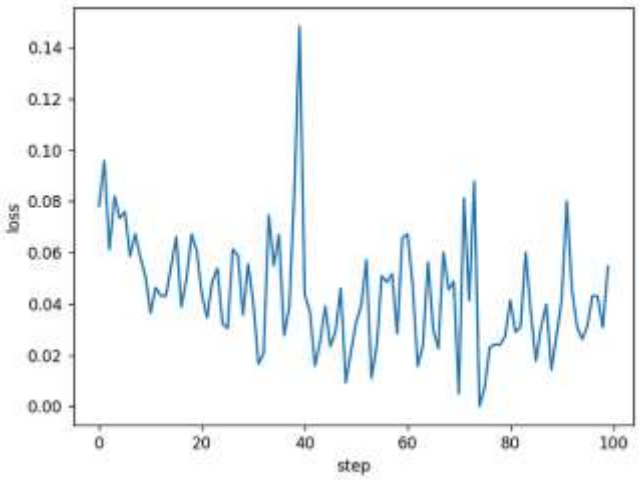
9. Training completion: After completing the training loop, the total training time (`training_time`) is calculated by subtracting the start time from the current time. The final loss values are displayed, and if a `saved_model` path is provided, the trained model, along with the losses and start time, is saved using the `save_model()` function.

The training process trains the LSTM-based speaker encoder by optimizing the model parameters with respect to the triplet loss. The model iteratively learns to discriminate between different speakers and generate speaker embeddings that capture the speaker characteristics present in the training data.

During model training, we can enhance the efficiency of our code by leveraging multithreading, which allows for parallel execution of certain operations and can significantly speed up the training process. Additionally, after the training is completed, we can visualize the performance of our model by plotting a graph of

the loss versus the number of training epochs. This graph provides valuable insights into the model's convergence and helps us assess the effectiveness of our training procedure.

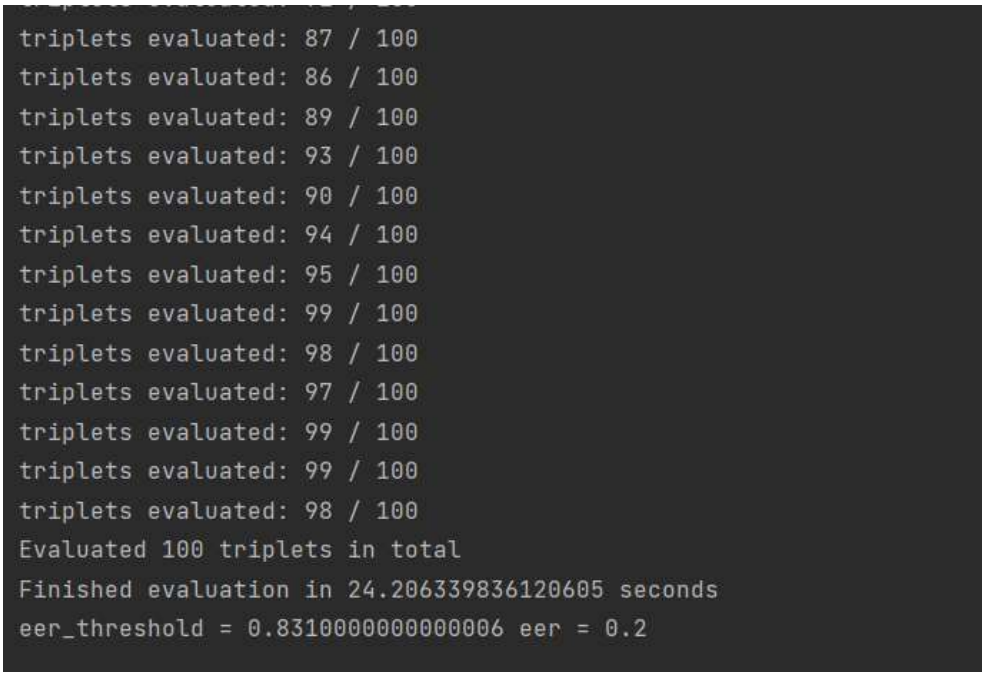
```
def run_training():
    print("Training data:", myconfig.TRAIN_DATA_DIR)
    speaker_to_utterance =
dataset.get_librispeech_speaker_to_utterance(myconfig.TRAIN_DATA_DIR)

    with multiprocessing.Pool(myconfig.NUM_PROCESSES) as pool:
        losses = train_network(speaker_to_utterance,
                               myconfig.TRAINING_STEPS,
                               myconfig.SAVED_MODEL_PATH,
                               pool)
    plt.plot(losses)
    plt.xlabel("step")
    plt.ylabel("loss")
    plt.show()
```



Graph of Loss vs Epoch (Image from Author)

Model Evaluation & Result

For model evaluation, we will use Equal Error Rate (EER), which is a common metric used in speaker recognition. It iterates over different threshold values to find the threshold that minimizes the difference between false acceptance rate (FAR) and false rejection rate (FRR).



The output of Evaluation.py (Image by Author)

In the above image, `eer_threshold` represents the threshold value at which the Equal Error Rate (EER) is achieved. It indicates the similarity score threshold that balances the false acceptance and false rejection rates. In the example, the `eer_threshold` is 0.8310000000000006. On the other hand, `eer` represents the EER itself, which is the average of the false acceptance rate and the false rejection rate. In the example, the EER is 0.2, indicating a 20% error rate in both false acceptances and false rejections.

These values provide insights into the performance of the speaker recognition system. The `eer_threshold` helps determine the threshold at which the system achieves a balanced error rate, while the `eer` gives an overall measure of the system's accuracy in distinguishing between genuine and impostor samples.

Difference between Model Training and Model Evaluation method

During training, the model is trained using triplets of data consisting of an anchor, a positive segment, and a negative segment. These triplets are used to learn embeddings that can discriminate between different speakers. In each batch

during training, the `batch_input` contains multiple triplets, and the model is trained to optimize the embedding space based on the relationships between the anchor, positive, and negative segments within each triplet.

However, during testing or evaluation, the goal is to compute embeddings for individual utterances and compare them to determine the similarity between different speakers. In this case, there is no need to compute embeddings for triplets because there are no anchor-positive-negative relationships to consider. Instead, each utterance is processed independently to obtain its embedding.

In summary, during training, triplets of data are used to train the model, while during testing/evaluation, individual utterances are processed separately to compute embeddings and perform similarity comparisons.

Conclusion

In conclusion, while our speaker recognition model based on LSTM architecture has shown promising results, there are opportunities for further improvement. Exploring the use of Transformer models, known for their effectiveness in natural language processing tasks, could enhance the model's ability to capture complex patterns in audio data. Refining feature extraction techniques, leveraging diverse datasets, and addressing challenges such as varying acoustic conditions and speaker characteristics are key areas for future research. By embracing advancements and collaboration, we can advance the accuracy and reliability of speaker recognition systems and contribute to this dynamic field.

To delve deeper into the code and explore additional possibilities, please visit my GitHub repository.

Thank you for joining us on this journey through the world of speaker recognition. Let us continue to push the boundaries and unlock the potential of this exciting field.