Conversational AI: Speech Recognition

Introduction

In today's world, we are increasingly surrounded by screens, with everyday objects being redesigned to include Wi-Fi and touchscreens. This raises concerns about screen overload. Voice interfaces offer a promising solution to reduce reliance on screens.

However, for independent developers and entrepreneurs, creating a straightforward speech recognition system using free, open data can be challenging. Many voice recognition datasets require extensive preprocessing before they can be used in machine learning models.

To address this, TensorFlow has released the Speech Commands Dataset, which contains 65,000 one-second utterances of 30 short words spoken by thousands of individuals. This dataset enables developers to build algorithms that recognize simple spoken commands. By enhancing the accuracy of open-source voice recognition tools, we can make voice interfaces more effective and accessible to a wider audience.

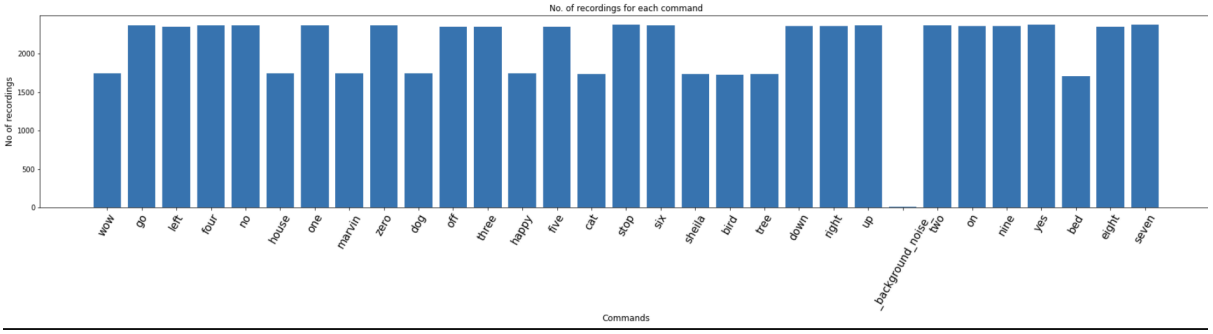
Many voice interfaces use keyword spotting to initiate interactions, like saying "Hey Google" or "Hey Siri." Once the device detects this, it sends the audio to a cloud service for processing. However, continuously sending audio from all devices to the cloud is impractical due to high costs and privacy concerns.

Instead, devices run a local recognition module that listens for trigger phrases. When a trigger is detected, the audio is sent to the cloud for further processing. Local models must be efficient due to limited processing power and battery life on mobile devices. They also need to minimize false positives, focus on recognizing short phrases, and provide quick feedback, even if the full response from the server takes time.

This makes keyword spotting more resource-constrained than server-based speech recognition, requiring smaller, energy-efficient models designed to work with limited computing power.

The dataset initially included digits from zero to nine and ten common command words useful for IoT and robotics: "Yes", "No", "Up", "Down", "Left", "Right", "On", "Off", "Stop", and "Go." In version 2, four more commands were added: "Backward", "Forward", "Follow", and "Learn."

One of the key challenges in keyword recognition is filtering out speech that doesn’t contain trigger words. To address this, additional words were included to test the model’s ability to distinguish non-trigger words. Some, like “Tree,” were chosen because they sound similar to target words, while others were selected randomly to cover a range of phonemes. This final set included: "Bed", "Bird", "Cat", "Dog", "Happy", "House", "Marvin", "Sheila", "Tree", and "Wow."



Implementation

Preprocessing the audio waves

In the data exploration part earlier, we have seen that the duration of a few recordings is less than 1 second and the sampling rate is too high. So, let us read the audio waves and use the below-preprocessing steps to deal with this.

Here are the two steps we'll follow:

Resampling

Removing shorter commands of less than 1 second

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 8000, 1)]         0
_________________________________________________________________
conv1d (Conv1D)              (None, 7988, 8)           112
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 2662, 8)           0
_________________________________________________________________
dropout (Dropout)            (None, 2662, 8)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 2652, 16)          1424
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 884, 16)           0
_________________________________________________________________
dropout_1 (Dropout)          (None, 884, 16)           0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 876, 32)           4640
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 292, 32)           0
_________________________________________________________________
dropout_2 (Dropout)          (None, 292, 32)           0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 286, 64)           14400
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 95, 64)            0
_________________________________________________________________
dropout_3 (Dropout)          (None, 95, 64)            0
_________________________________________________________________
flatten (Flatten)            (None, 6080)              0
_________________________________________________________________
dense (Dense)                (None, 256)               1556736
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,611,498
Trainable params: 1,611,498
Non-trainable params: 0
```

For the ease of computation only 10 keywords are used to train and test data this could be well increased to all keywords provided we have enough resources and time.

This model uses a 1D Convolutional Neural Network (CNN) for sequential data processing. It consists of four Conv1D layers with max pooling and dropout for regularization. The data is flattened after feature extraction, followed by dense layers for classification. The final dense layer outputs 10 classes, with 1.6 million trainable parameters.

Input Layer (InputLayer):

Specifies the shape and structure of the input data. In this case, the input is a sequence of length 8000 with 1 feature per time step.

Conv1D (1D Convolutional Layer):

This layer applies convolutional filters over one-dimensional data (e.g., time series, sequences). It learns to extract local features by sliding small filters across the input. Each Conv1D layer adds filters to capture increasingly complex patterns.

MaxPooling1D (1D Max Pooling Layer):

This layer reduces the dimensionality of the feature map by taking the maximum value from each window of data, helping to downsample the input. MaxPooling helps reduce computational load and focus on the most prominent features.

Dropout:

Dropout is a regularization technique that randomly deactivates a percentage of neurons during training. This prevents the model from becoming too dependent on specific neurons and helps reduce overfitting.

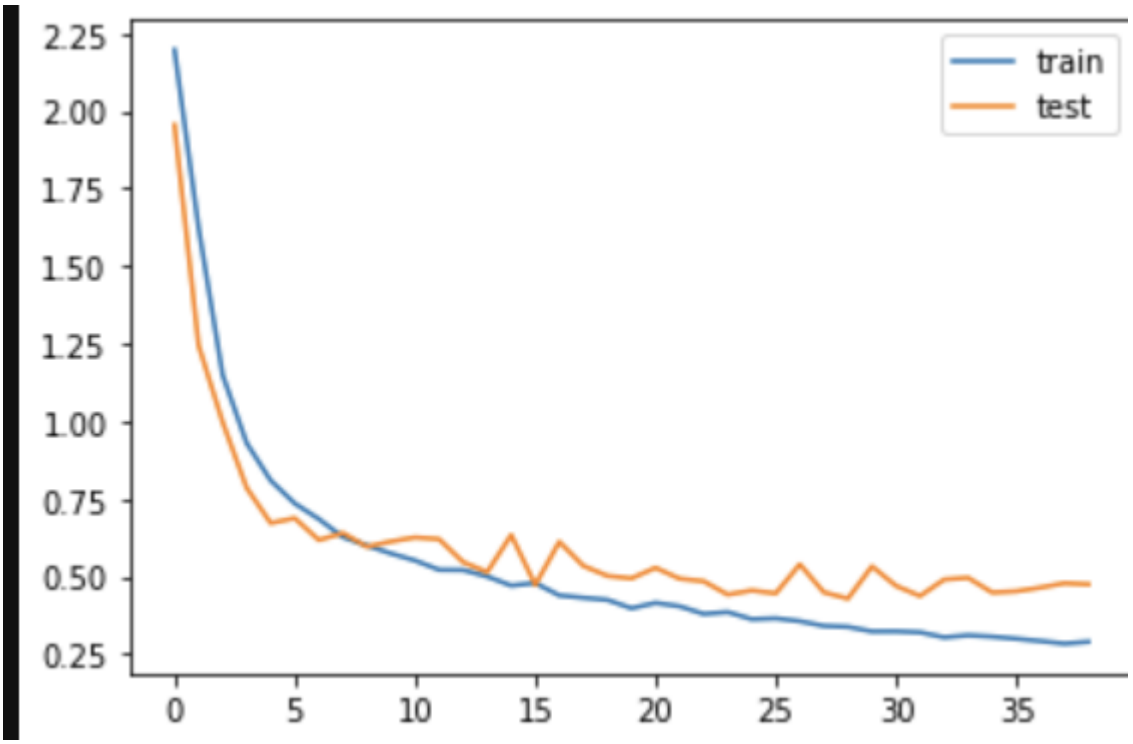
Flatten:

This layer reshapes the multi-dimensional output of the previous layers into a single vector. It prepares the data for the fully connected layers by collapsing the spatial structure of the data.

Dense (Fully Connected Layer):

Each neuron in a dense layer is connected to every neuron in the previous layer. Dense layers are used for classification or regression, typically toward the end of the model. In this model, dense layers are used to make the final predictions after feature extraction by the convolutional layers.

These layers together create a pipeline that extracts features (using Conv1D and MaxPooling), regularizes the model (using Dropout), and finally classifies the input (using Dense layers).



Model “Accuracy” based on train and validation dataset

Accuracy is used as a testing criterion since it is a classification dataset and our main task was to spot keyword and further convert to text.

Future Scope

For the future scope to use it for our own recording, the Python file also consists of testing model capability with your own voice by doing the following steps:
1. Recording Audio
The code uses the sounddevice (as sd) library to record audio.
The sample rate is set to 16000 samples per second.

Duration is set to 1 second.
Audio is recorded using the sd. rec() function, which captures audio data with the specified sample rate, duration, and single channel (mono).
The audio recording is saved to the file 'yes.wav' using the soundfile library (sf.write()).
2. Loading and Resampling Audio
The code then loads an existing audio file ('stop.wav') using the librosa.load() function, which reads the audio at a sample rate of 16000.
After loading, the audio is resampled from 16000 Hz to 8000 Hz using librosa.resample() to match the model's expected input.
Finally, the resampled audio is played back using the IPython.display.Audio.