Speech Commands:
A Dataset for Limited -
Vocabulary Speech Recognition

Paper : https://arxiv.org/abs/1804.03209

Summary : The paper presents the "Speech Commands" dataset, a specialized audio collection for training and evaluating keyword spotting models. It contains over 105,000 utterances of 35 words from 2,618 speakers, designed for tasks like on-device recognition of keywords (e.g., "Yes", "No", "Stop") in low-resource environments. The dataset addresses challenges like minimizing false positives from background noise and ensuring energy-efficient models for devices with limited computational power. It is intended to standardize testing protocols by providing a benchmark for reproducible and comparable accuracy metrics, with a focus on tasks distinct from general speech recognition, such as keyword detection.

Dataset Analysis

- **Preprocessing** : The code resamples the audio files and normalizes them to ensure consistent input for the classifier. Silence removal and basic noise reduction are implemented.
- **Feature Extraction** : The code extracts MFCCs (Mel-Frequency Cepstral Coefficients) from the audio files, which are used as input features for the classifier.

```python
import os
import tarfile
import urllib.request

DATASET_URL = "http://download.tensorflow.org/data/speech_commands_v0.02.tar.gz"
data_dir = './data'

if not os.path.exists(data_dir):
    os.makedirs(data_dir)


dataset_path = os.path.join(data_dir, 'speech_commands_v0.02.tar.gz')
urllib.request.urlretrieve(DATASET_URL, dataset_path)


with tarfile.open(dataset_path, 'r:gz') as tar:
    tar.extractall(path=data_dir)

print("Dataset downloaded and extracted.")
```

```
Dataset downloaded and extracted.
```

```python
def decode_audio(audio_binary):
    audio, _ = tf.audio.decode_wav(audio_binary)
    return tf.squeeze(audio, axis=-1)

def get_label(file_path):
    parts = tf.strings.split(file_path, os.path.sep)
    return parts[-2]

def get_waveform_and_label(file_path):
    audio_binary = tf.io.read_file(file_path)
    waveform = decode_audio(audio_binary)
    label = get_label(file_path)
    return waveform, label


files = tf.io.gfile.glob(str(data_dir) + '/*/*.wav')
files = tf.random.shuffle(files)

print("Number of audio files:", len(files))


files_ds = tf.data.Dataset.from_tensor_slices(files)
waveform_ds = files_ds.map(get_waveform_and_label, num_parallel_calls=AUTOTUNE)
```

```
Number of audio files: 105835
```

Classifier Training

- **Model Architecture** : A Convolutional Neural Network (CNN) is used, consisting of several convolutional and pooling layers, followed by fully connected layers.
- **Training** : The model is trained using the categorical cross-entropy loss function and Stochastic Gradient Descent (SGD) optimizer. Dropout is applied as a regularization technique to prevent overfitting.
- **Early Stopping** : The code implements early stopping to halt training when the validation loss stops improving.

```python
data_dir = pathlib.Path("./data")

commands = np.array(tf.io.gfile.listdir(str(data_dir)))
commands = commands[commands != 'README.md']
print('Commands:', commands)
```

```
Commands: ['forward' 'right' 'yes' 'no' 'left' '.DS_Store' 'off' 'happy' 'learn'
 'four' 'sheila' 'marvin' 'six' 'speech_commands_v0.02.tar.gz' 'stop'
 'nine' 'LICENSE' 'tree' 'two' 'up' 'dog' 'on' 'go' 'one'
 'validation_list.txt' 'down' 'follow' 'eight' 'three' 'zero'
 '_background_noise_' 'testing_list.txt' 'bird' 'seven' 'backward' 'house'
 'five' 'cat' 'wow' 'bed' 'visual']
```

```python
def preprocess_dataset(files):
    files_ds = tf.data.Dataset.from_tensor_slices(files)
    output_ds = files_ds.map(get_waveform_and_label, num_parallel_calls=AUTOTUNE)
    output_ds = output_ds.map(get_spectrogram_and_label_id, num_parallel_calls=AUTOTUNE)
    return output_ds

total_files = len(files)
train_size = int(0.8 * total_files)
val_size = int(0.1 * total_files)
test_size = total_files - train_size - val_size

train_files = files[:train_size]
val_files = files[train_size:train_size+val_size]
test_files = files[train_size+val_size:]

print(f"Total files: {total_files}")
print(f"Train files: {len(train_files)}")
print(f"Validation files: {len(val_files)}")
print(f"Test files: {len(test_files)}")

# Create datasets
train_ds = preprocess_dataset(train_files)
val_ds = preprocess_dataset(val_files)
test_ds = preprocess_dataset(test_files)

# Batch the datasets
batch_size = 64
train_ds = train_ds.batch(batch_size)
val_ds = val_ds.batch(batch_size)
test_ds = test_ds.batch(batch_size)

# Use buffered prefetching
train_ds = train_ds.cache().prefetch(AUTOTUNE)
val_ds = val_ds.cache().prefetch(AUTOTUNE)
test_ds = test_ds.cache().prefetch(AUTOTUNE)
# train_files = files[:int(len(files) * 0.8)]
# val_files = files[int(len(files) * 0.8):]
# test_files = files[int(len(files) * 0.2):]

# train_ds = preprocess_dataset(train_files)
# val_ds = preprocess_dataset(val_files)
# test_ds= preprocess_dataset(test_files)

# batch_size = 64
# train_ds = train_ds.batch(batch_size)
# val_ds = val_ds.batch(batch_size)
# test_ds = test_ds.batch(batch_size)
```

```
Total files: 105835
Train files: 84668
Validation files: 10583
Test files: 10584
```

Performance Results using Standard Benchmarks

- **Metrics**: The model's accuracy is computed to be 0.86, along with precision, recall, and F1-score for evaluating the classifier's effectiveness.
- **Confusion Matrix** : A confusion matrix is generated to analyze misclassifications, providing insights into where the model may be making errors.

```
EPOCHS = 5
history = model.fit(
    train_ds.cache().prefetch(buffer_size=tf.data.AUTOTUNE),
    validation_data=val_ds.cache().prefetch(buffer_size=tf.data.AUTOTUNE),
    epochs=EPOCHS
)
```

```
Epoch 1/5
1323/1323 ━━━━━━━━━━━━━━━━━━━━ 190s 138ms/step - accuracy: 0.5146 - loss: 1.7881 - val_accuracy: 0.8224 - val_loss: 0.6301
Epoch 2/5
1323/1323 ━━━━━━━━━━━━━━━━━━━━ 37s 28ms/step - accuracy: 0.8530 - loss: 0.5056 - val_accuracy: 0.8514 - val_loss: 0.5252
Epoch 3/5
1323/1323 ━━━━━━━━━━━━━━━━━━━━ 37s 28ms/step - accuracy: 0.9068 - loss: 0.3150 - val_accuracy: 0.8521 - val_loss: 0.5696
Epoch 4/5
1323/1323 ━━━━━━━━━━━━━━━━━━━━ 37s 28ms/step - accuracy: 0.9331 - loss: 0.2198 - val_accuracy: 0.8643 - val_loss: 0.5996
Epoch 5/5
1323/1323 ━━━━━━━━━━━━━━━━━━━━ 42s 28ms/step - accuracy: 0.9495 - loss: 0.1649 - val_accuracy: 0.8640 - val_loss: 0.6647
```

```
model.save('speech_recognition_model.keras')
```

```
test_loss, test_accuracy = model.evaluate(test_ds)
print(f"Test accuracy: {test_accuracy:.2f}")
```

```
166/166 ━━━━━━━━━━━━━━━━━━━━ 24s 144ms/step - accuracy: 0.8548 - loss: 0.6881
Test accuracy: 0.86
```