Speech Classification

By: Rohit Rana

Roll.no: 102116012

Section: 4CS09

To: Dr. Raghav B. Venkataramaiyer

Task Description

The objective of this lab evaluation is to work with the Speech Commands dataset, as detailed in the paper *An Overview of the Speech Commands Dataset*. The tasks include summarizing the paper, performing statistical analysis on the dataset, training a classifier, and fine-tuning it with custom-recorded samples.

Task Breakdown

1. Read and Summarize the Paper
2. Download and Analyze the Dataset
3. Train a Classifier (Google Collab in the Repo)
4. Report Performance Results
5. Create a New Dataset with Custom Samples (google link attached)
6. Fine-tune the Classifier (To be Done)
7. Report the Results

Summary of the Paper (50 Words)

The paper "Speech Command Recognition with TensorFlow" presents a simple yet effective approach for recognizing spoken commands using a neural network model. It introduces a dataset of one-second long audio recordings of 30 different commands. The authors build a small convolutional neural network (CNN) to classify these commands and discuss performance benchmarks on recognition accuracy and model efficiency.

Creating a Custom Dataset.

```python
import sounddevice as sd
from scipy.io.wavfile import write
import os

fs = 44100

duration = 1

words =['backward', 'bed', 'bird', 'cat', 'dog', 'down', 'eight','five', 'follow', 'forward', 'four', 'go',
 'happy', 'house', 'learn', 'left', 'marvin', 'nine', 'no', 'off', 'on', 'one', 'right', 'seven', 'sheila', 'six',
 'stop', 'three', 'tree', 'two', 'up', 'visual', 'wow', 'yes', 'zero']

for word in words:
    output_folder = f"{word}_recordings"

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

    for i in range(5):
        print(f"Recording {i+1} of 5 for the word '{word}'... Please say the word '{word}'")

        audio_data = sd.rec(int(duration * fs), samplerate=fs, channels=2)
        sd.wait()
        print(f"Recording {i+1} for '{word}' finished!")

        filename = os.path.join(output_folder, f"{word}_recording_{i+1}.wav")
        write(filename, fs, audio_data)
        print(f"Audio saved as {filename}")

print("All recordings complete!")
```

Link to the Dataset in the README file of the git repository.

1. Dataset Handling

SpeechCommands Dataset

The code begins by defining a custom subclass, `SubsetSC`, to load different subsets (training, validation, and testing) of the **SpeechCommands** dataset. The dataset contains labeled audio clips of spoken commands.

- **Training and Testing Datasets**:
  - A `train_set` for training and a `test_set` for testing are initialized using the `SubsetSC` class.
  - The dataset provides waveforms, sample rates, labels, and additional metadata.

Custom Audio Dataset

In addition to SpeechCommands, a custom audio dataset stored on Google Drive is also loaded using a class `AudioDataset`, which handles:

- Loading `.wav` files,
- Resampling to a common frequency (16 kHz),
- Returning waveforms, labels, and other metadata.

2. Data Preprocessing

- **Resampling**: The audio data from the SpeechCommands dataset is resampled to 8 kHz, while the custom dataset is resampled to 16 kHz.
- **Batch Collation**:
  - `collate_fn` handles padding sequences of different lengths and collating them into tensors.
  - The `pad_sequence` function is used to pad audio sequences to a uniform length, ensuring they can be batched together during training.

3. Exploratory Data Analysis

The code performs some exploratory data analysis:

- **Visualizing Waveforms**: Random audio samples are plotted, showing the waveform amplitude over time.
- **Statistics of Waveforms**: Histograms for the mean, standard deviation, minimum, and maximum values of waveforms across the training dataset are computed and plotted.
- **Per-Speaker Statistics**: Mean and standard deviation of the waveforms are aggregated per speaker, helping to understand variability across speakers.

4. Model Architecture

The model is based on a convolutional neural network (CNN) architecture called **M5**, which is suitable for audio data:

- **Convolutional Layers**: Four 1D convolutional layers with batch normalization and ReLU activation are used.
- **Pooling**: Max-pooling layers are applied after convolution to reduce the temporal dimension.
- **Fully Connected Layer**: A final fully connected layer maps the output to 35 classes (one for each command label in SpeechCommands).

The network uses about 0.5 million trainable parameters.

5. Training

The model is trained using:

- **Adam Optimizer**: Optimizes the model's weights with a learning rate of 0.01 and weight decay of 0.0001.
- **Learning Rate Scheduler**: Reduces the learning rate every 20 epochs by a factor of 0.1 to fine-tune the learning process.
- **Loss Function**: Negative log-likelihood loss (`nll_loss`) is used for training.

The model's parameters are updated through backpropagation, and the loss is computed at regular intervals (every 20 batches) to monitor training performance.

6. Evaluation

- **Accuracy**: After training, the model's performance is evaluated on the test set by calculating the prediction accuracy.
- **Prediction Pipeline**: The `predict` function performs inference on unseen data, returning the predicted class label for a given audio waveform.

7. Training with Custom Dataset

- A custom dataset of 30 samples per 35 classes is loaded from the drive.
- **Data Splitting**: The custom dataset is split into an 80% training set and 20% test set.
- The training and testing pipelines are reused to train the model on this new dataset, with the model achieving a **46.9% accuracy** on the first round of training.

8. Saving and Loading the Model

The model state, along with optimizer and scheduler states, is saved to a checkpoint file (`model_checkpoint.pth`). This ensures that the training process can be resumed later.

Similarly, after fine-tuning the model with the custom dataset, the updated model is saved to another checkpoint file (`CustomDataset_model_checkpoint.pth`).

9. Real-Time Testing

The model's real-time capabilities are tested by loading an audio file and predicting the spoken command:

- The user can record their own audio and test the model's prediction in real-time, which is useful for practical deployment scenarios.

10. Performance and Results

The model achieved moderate performance on the SpeechCommands dataset and custom dataset. After 3 epochs of training on the custom dataset, a significant number of incorrect predictions were observed, suggesting that further tuning and data augmentation might be necessary to improve accuracy.

Outcome:

1. **Customizable Pipeline**: The code demonstrates a complete, customizable pipeline for training, evaluating, and saving an audio classification model.
2. **Real-time Inference**: The ability to predict spoken commands in real-time using recorded audio makes the model practical for deployment in interactive systems.
3. **Moderate Accuracy**: The current model shows moderate accuracy (46.9%) on a custom dataset, indicating that more training or architectural adjustments might be needed for better performance.