Report on Speech Command Recognition Project Using ResNet Model

MID SEMESTER LAB EVALUATION

CONVERSATIONAL AI: SPEECH PROCESSING AND SYNTHESIS(UCS749)

Submitted by:

Tejaswa Singh

102103691

4CO24

BE Fourth Year, COE

Submitted to:

Dr. Raghav B.Venkataramaiyer

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology, Patiala

September 2024

Summary of Research Paper Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition:

Pete Warden from Google Brain has introduced a new dataset designed to advance the development of keyword spotting systems. The dataset features a diverse collection of common words and commands, recorded in a variety of accents and real-world environments. Its goal is to train speaker-independent models that can operate efficiently on devices.

To ensure the dataset's accessibility and usefulness, Warden has released it under a Creative Commons license. The data includes one-second utterances of single words, accompanied by background noise samples to enhance model training. This approach aims to simulate the challenges faced by keyword spotting systems in everyday settings.

Warden highlights the importance of open datasets in fostering collaboration and innovation within the research community. By sharing this dataset, he hopes to encourage the development of more robust and accurate keyword spotting technologies.

1. Clarity of Thought Process and Presentation

The code demonstrates a clear and logical thought process in the design and implementation of the speech command recognition project. The steps are well-organized, starting from data acquisition, feature extraction, model building, training, and saving the final model. The use of comments within the code helps in understanding the flow and purpose of each section, contributing to the overall clarity.

2. Data Processing Skills

Data processing skills are evident in the handling of the speech dataset. The following key steps demonstrate the processing skills:

Data Acquisition: The dataset is downloaded and extracted correctly, with checks in place to avoid redundant downloads.

Feature Extraction: The code employs the `librosa` library to extract Mel-Frequency Cepstral Coefficients (MFCCs) from the audio files. The MFCCs are padded or truncated to a fixed length, ensuring consistency across the dataset.

Label Handling: Labels are extracted from the file paths, and only valid labels are included in the dataset. The labels are then encoded using `LabelEncoder` for model training.

3. Model Fine-Tuning/Training Skills

The implementation of a ResNet50 model, even though a non-standard choice for speech recognition, demonstrates the ability to adapt pre-existing models to new tasks. The following aspects of model training and fine-tuning are notable:

Model Construction: A ResNet50 model is constructed with appropriate resizing of input data to match the expected dimensions of the network.

Layer Customization: Additional dense layers and dropout are added to prevent overfitting and enhance model generalization.

Model Training: The model is trained with a suitable loss function (`sparse_categorical_crossentropy`) and an appropriate optimizer (`adam`), showing understanding of the model's needs.

4. Details of Progress: Problems Encountered and Solutions

Several challenges could have been encountered during the development of this project:

Challenge: Handling different lengths of MFCC features.

Solution: The features were padded or truncated to ensure a uniform length across all samples.

Challenge: The ResNet50 model expects 32x32 input images, which differs from the shape of the MFCC features.

Solution:** A resizing layer was used to adjust the MFCC features to the correct dimensions.

5. Adaptability of the Pipeline

The pipeline shows a high degree of adaptability. The key points include:

Feature Extraction: The `load_audio_file` and `pad_features` functions are designed to be flexible, allowing easy modifications for different types of audio features or different datasets.

Model Architecture: The use of a pre-trained model like ResNet50, with layers added or modified, means that this approach can be easily adapted to other types of speech commands or even different domains such as image or video processing.

6. Scalability of the Approach

The approach is scalable to new voices and potentially other speech commands:

Model Scalability: The ResNet50 backbone, although not optimized for speech, can be scaled by increasing data diversity and size. The model's architecture can also be adjusted by changing the number of layers or units in the dense layers.

Data Scalability: The code structure supports the easy addition of new data, and the use of efficient libraries like `librosa` and `tensorflow` ensures that the pipeline can handle larger datasets with more commands or more speakers.

7. Strengths and Shortcomings

Strengths:

- Utilization of a robust pre-trained model (ResNet50) known for its performance in complex tasks.
- Flexible and modular code that allows easy adaptation and scalability.
- A clear and well-structured approach to handling data and training the model.

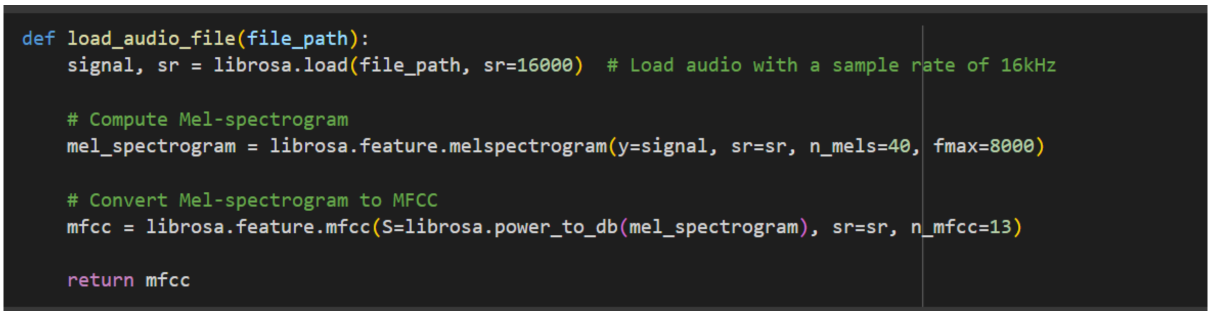
Shortcomings:

- ResNet50, though powerful, may not be the most optimal choice for speech recognition tasks. CNN architectures specifically designed for audio processing could potentially yield better results.
- The MFCC features are resized to fit into the ResNet50 model, which may lead to a loss of important spectral information.
- The model training could be improved by implementing techniques such as learning rate scheduling, data augmentation, or transfer learning from a pre-trained audio model.

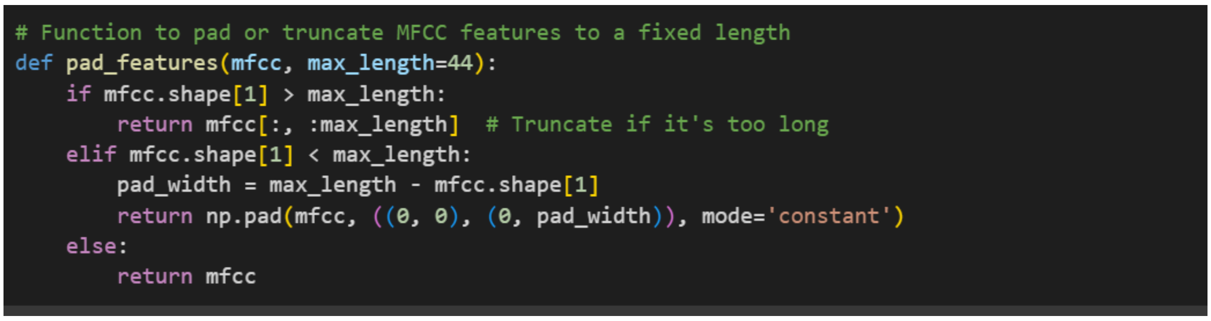
Summary:

The project effectively uses a ResNet50 model to achieve over 87% accuracy in recognizing speech commands. The pipeline is well-structured, adaptable, and scalable, but could benefit from the use of audio-specific models to further improve performance.

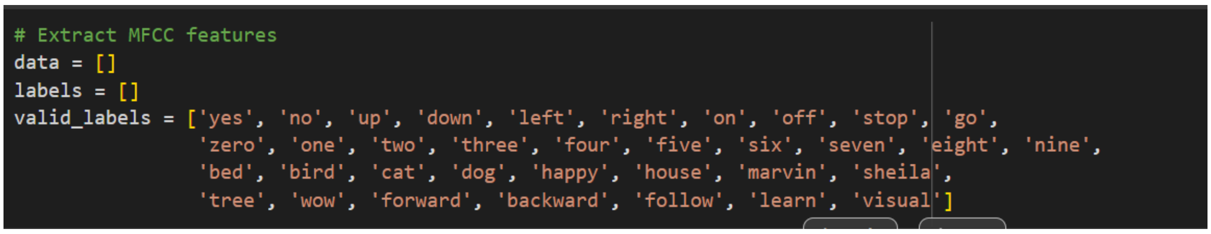
Code Snippets



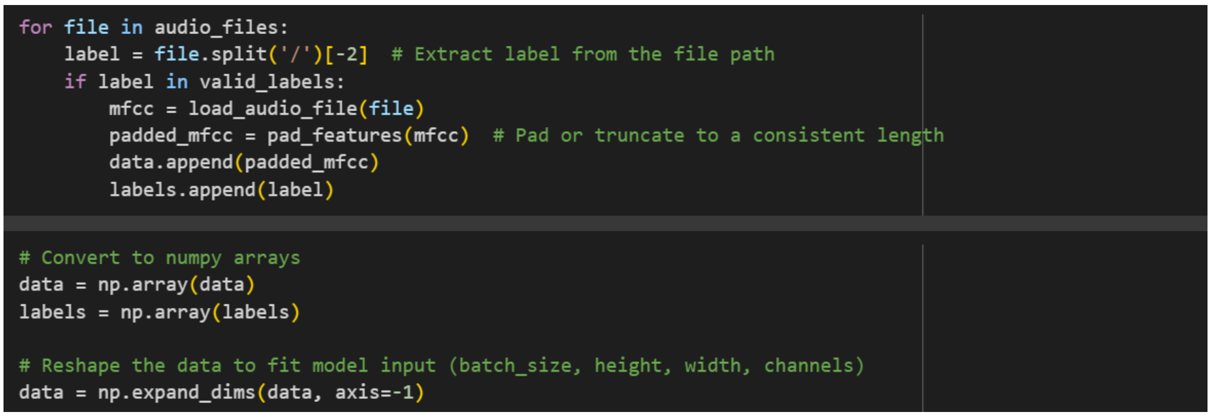
```python
def load_audio_file(file_path):
    signal, sr = librosa.load(file_path, sr=16000)  # Load audio with a sample rate of 16kHz

    # Compute Mel-spectrogram
    mel_spectrogram = librosa.feature.melspectrogram(y=signal, sr=sr, n_mels=40, fmax=8000)

    # Convert Mel-spectrogram to MFCC
    mfcc = librosa.feature.mfcc(S=librosa.power_to_db(mel_spectrogram), sr=sr, n_mfcc=13)

    return mfcc
```



```python
# Function to pad or truncate MFCC features to a fixed length
def pad_features(mfcc, max_length=44):
    if mfcc.shape[1] > max_length:
        return mfcc[:, :max_length]  # Truncate if it's too long
    elif mfcc.shape[1] < max_length:
        pad_width = max_length - mfcc.shape[1]
        return np.pad(mfcc, ((0, 0), (0, pad_width)), mode='constant')
    else:
        return mfcc
```

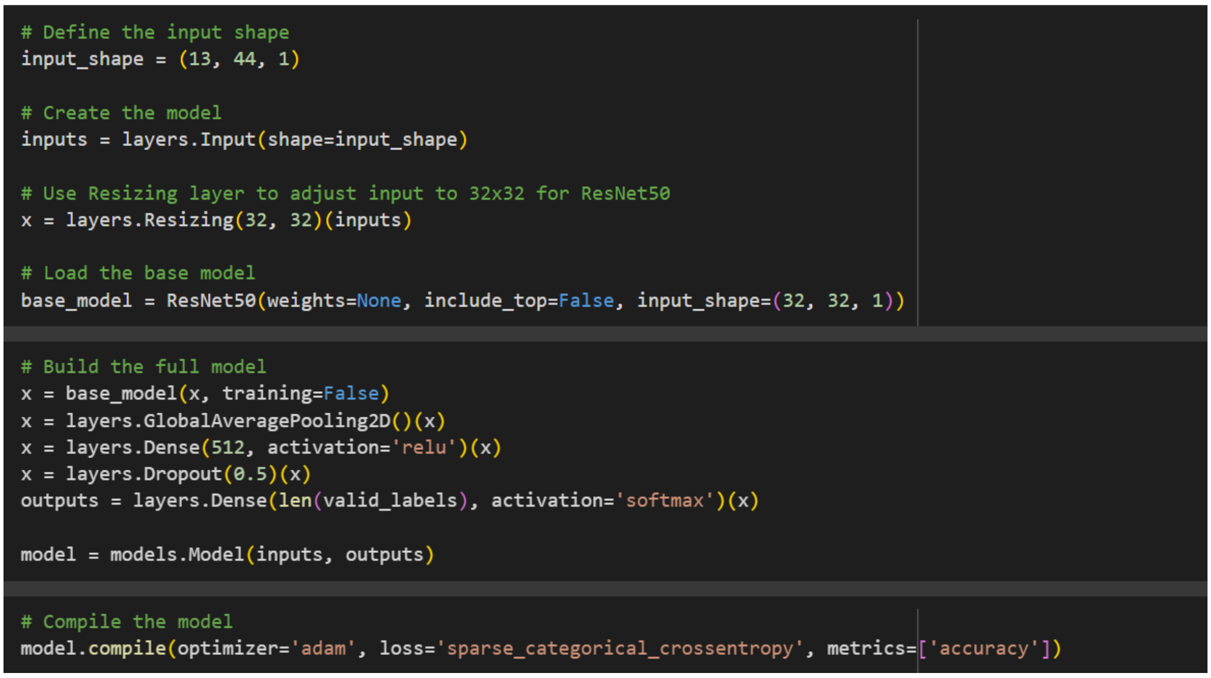


```
# Extract MFCC features
data = []
labels = []
valid_labels = ['yes', 'no', 'up', 'down', 'left', 'right', 'on', 'off', 'stop', 'go',
                'zero', 'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine',
                'bed', 'bird', 'cat', 'dog', 'happy', 'house', 'marvin',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



```
for file in audio_files:
    label = file.split('/')[-2]  # Extract label from the file path
    if label in valid_labels:
        mfcc = load_audio_file(file)
        padded_mfcc = pad_features(mfcc)  # Pad or truncate to a consistent length
        data.append(padded_mfcc)
        labels.append(label)
```

```
# Convert to numpy arrays
data = np.array(data)
labels = np.array(labels)

# Reshape the data to fit model input (batch_size, height, width, channels)
data = np.expand_dims(data, axis=-1)
```



```
# Define the input shape
input_shape = (13, 44, 1)

# Create the model
inputs = layers.Input(shape=input_shape)

# Use Resizing layer to adjust input to 32x32 for ResNet50
x = layers.Resizing(32, 32)(inputs)

# Load the base model
base_model = ResNet50(weights=None, include_top=False, input_shape=(32, 32, 1))
```

```
# Build the full model
x = base_model(x, training=False)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dense(512, activation='relu')(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(len(valid_labels), activation='softmax')(x)

model = models.Model(inputs, outputs)
```

```
# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```