Conversational AI: Speech recognition report

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology

(Deemed to be University), Patiala – 147004

Submitted By:

Mohammed Ayesha Sami (102117110)

Submitted To:

B.V Raghav

INTRODUCTION

This report provides a comprehensive guide on how to load, preprocess, and train a machine learning model using the Speech Commands dataset, as well as how to fine-tune the model on your own dataset. The Speech Commands dataset consists of audio recordings of spoken words that are used to train models for speech recognition tasks.

Step 1: Download and Extract the Dataset

To begin, download the Speech Commands dataset and extract it into a directory.

```
import os
import tarfile
import numpy as np
import tensorflow as tf  # Import TensorFlow

tar_file_path = 'dataset.tar.gz'  # Replace with the path to your tar.gz file
extracted_dir = 'speech_commands_dataset'  # Directory to extract to

# Check if the directory already exists to avoid re-extracting
if not os.path.exists(extracted_dir):
    os.makedirs(extracted_dir)  # Create the directory if it doesn't exist
    with tarfile.open(tar_file_path, 'r:gz') as tar:
        tar.extractall(extracted_dir)
        print(f"Extracted to {extracted_dir}")

# List the contents of the extracted folder
commands = np.array(tf.io.gfile.listdir(extracted_dir))
commands = commands[(commands != 'README.md') & (commands != '.DS_Store') & (commands != 'background_noise')]

print('Commands:', commands)
```

And list the contents of the extracted folder.

Step 2: Load the Dataset Using TensorFlow

Use TensorFlow's audio_dataset_from_directory method to create training and validation datasets from the extracted audio files.

```
data_dir = extracted_dir

# Load dataset
train_ds, val_ds = tf.keras.utils.audio_dataset_from_directory(
    directory=data_dir,
    batch_size=64,
    validation_split=0.2,
    seed=0,
    output_sequence_length=16000,
    subset='both')

label_names = np.array(train_ds.class_names)
print("Label names:", label_names)
```

Step 3: Preprocess the Audio Data

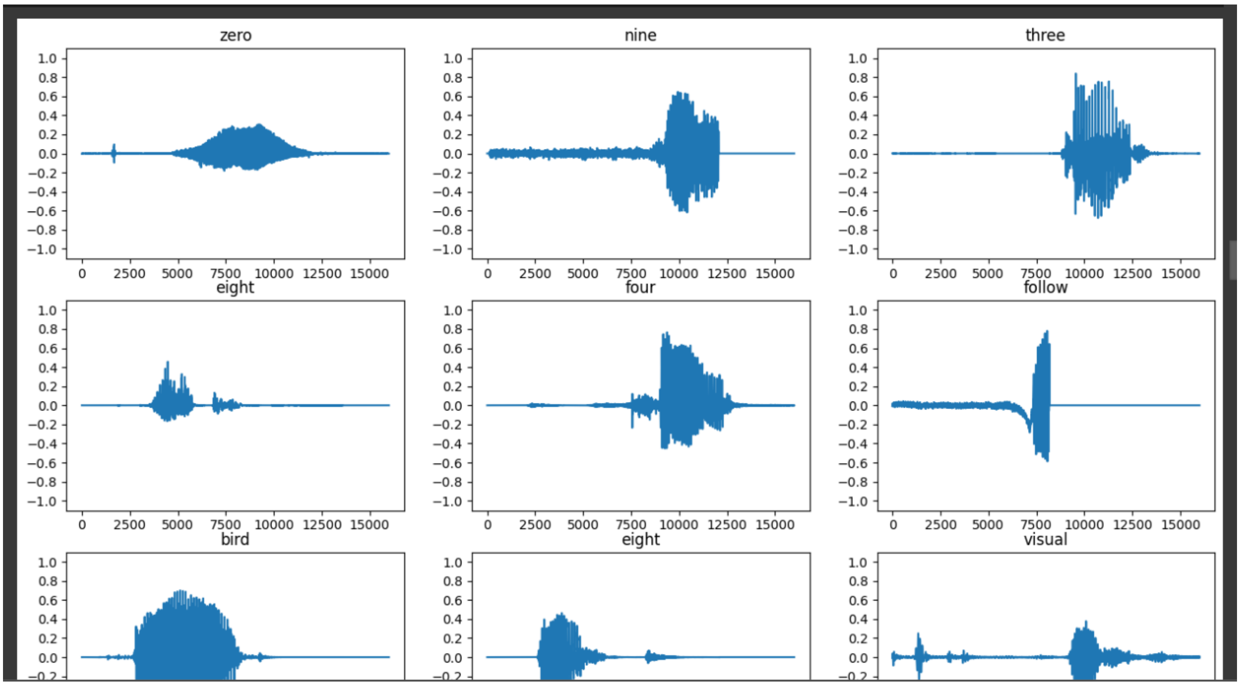
To prepare the data for training, we need to preprocess it by squeezing the audio tensors and splitting the validation set for testing.

```
def squeeze(audio, labels):
    audio = tf.squeeze(audio, axis=-1)
    return audio, labels

train_ds = train_ds.map(squeeze, tf.data.AUTOTUNE)
val_ds = val_ds.map(squeeze, tf.data.AUTOTUNE)

test_ds = val_ds.shard(num_shards=2, index=0)
val_ds = val_ds.shard(num_shards=2, index=1)
```

Step 5: Visualize Audio Waveforms

Visualize some example audio waveforms to understand the data.

```
plt.figure(figsize=(16, 10))
rows = 3
cols = 3
n = rows * cols
for i in range(n):
    plt.subplot(rows, cols, i+1)
    audio_signal = example_audio[i]
    plt.plot(audio_signal)
    plt.title(label_names[example_labels[i]])
    plt.yticks(np.arange(-1.2, 1.2, 0.2))
    plt.ylim([-1.1, 1.1])
plt.show()
```



Step 6: Convert Waveforms to Spectrograms

Convert the audio waveforms to spectrograms using Short-Time Fourier Transform (STFT), which are more suitable for input to neural networks.

```python
def get_spectrogram(waveform):
    # Convert the waveform to a spectrogram via a STFT.
    spectrogram = tf.signal.stft(
        waveform, frame_length=255, frame_step=128)
    # Obtain the magnitude of the STFT.
    spectrogram = tf.abs(spectrogram)
    # Add a `channels` dimension.
    spectrogram = spectrogram[..., tf.newaxis]
    return spectrogram

def plot_spectrogram(spectrogram, ax):
    if len(spectrogram.shape) > 2:
        assert len(spectrogram.shape) == 3
        spectrogram = np.squeeze(spectrogram, axis=-1)
    # Convert the frequencies to log scale and transpose.
    log_spec = np.log(spectrogram.T + np.finfo(float).eps)
    height = log_spec.shape[0]
    width = log_spec.shape[1]
    X = np.linspace(0, np.size(spectrogram), num=width, dtype=int)
    Y = range(height)
    ax.pcolormesh(X, Y, log_spec)

# Plot the waveform and its spectrogram
for example_audio, example_labels in train_ds.take(1):
    waveform = example_audio[0]
    spectrogram = get_spectrogram(waveform)

    fig, axes = plt.subplots(2, figsize=(12, 8))
    timescale = np.arange(waveform.shape[0])
    axes[0].plot(timescale, waveform.numpy())
    axes[0].set_title('Waveform')
    axes[0].set_xlim([0, 16000])

    plot_spectrogram(spectrogram.numpy(), axes[1])
    axes[1].set_title('Spectrogram')
    plt.suptitle(label_names[example_labels[0].numpy()])
    plt.show()
```

Step 7: Create TensorFlow Datasets for Spectrograms

Transform the datasets to use spectrograms instead of raw waveform

```
def make_spec_ds(ds):
  return ds.map(
      map_func=lambda audio,label: (get_spectrogram(audio), label),
      num_parallel_calls=tf.data.AUTOTUNE)

train_spectrogram_ds = make_spec_ds(train_ds)
val_spectrogram_ds = make_spec_ds(val_ds)
test_spectrogram_ds = make_spec_ds(test_ds)

for example_spectrograms, example_spect_labels in train_spectrogram_ds.take(1):
  break

rows = 7
cols = 3
n = rows*cols
fig, axes = plt.subplots(rows, cols, figsize=(16, 9))

for i in range(n):
    r = i // cols
    c = i % cols
    ax = axes[r][c]
    plot_spectrogram(example_spectrograms[i].numpy(), ax)
    ax.set_title(label_names[example_spect_labels[i].numpy()])

plt.show()
```

Step 8: Optimize Data Pipeline
Cache, shuffle, and prefetch data to optimize the input pipeline for training.

```
[13] train_spectrogram_ds = train_spectrogram_ds.cache().shuffle(10000).prefetch(tf.data.AUTOTUNE)
     val_spectrogram_ds = val_spectrogram_ds.cache().prefetch(tf.data.AUTOTUNE)
     test_spectrogram_ds = test_spectrogram_ds.cache().prefetch(tf.data.AUTOTUNE)
```

Step 9: Build and Compile the Model
Build a Convolutional Neural Network (CNN) model using Keras.

Step 10: Train the Model
Train the model using the spectrogram datasets.

Step 11: Evaluate the Model
Evaluate the trained model's performance on the test dataset.

Step 12: Plot Training History
Visualize the loss and accuracy over epochs to monitor the model's performance.

Step 13: Generate Predictions and Plot Confusion Matrix
Generate predictions on the test dataset and visualize the confusion matrix to understand the model's performance across different classes.