Project Report LE1

Conversational AI: Speech Processing and Synthesis

Submitted By:

Chhavi Gupta

Roll no: 102103322

Class: 4CO12

Submitted To:

Dr. BV Raghav

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology

(Deemed to be University), Patiala – 147004

Summary of Research Paper

The paper **"Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition"** provides a dataset designed for on-device keyword spotting to reduce network latency and dependence on cloud services. It enables developers to test models in a standardized environment, allowing for accurate comparisons using consistent metrics. The dataset focuses on speaker-independent commands captured through device microphones, with utterances restricted to one second. This co-design of machine learning and hardware supports reproducibility, helping teams benchmark models and improve top-one error rates in speech recognition systems.

Datasets Used

Google Speech Commands Dataset: The dataset contains 105,829 utterances of 35 words from 2,618 speakers, using hashed speaker IDs for privacy. It features a variety of background noise samples and supports automated and manual quality control methods.

84,668 files for training and 21,167 for validation. Larger dataset size offers better generalization and robustness to diverse audio inputs and background noise. The dataset is accessible for reproducible research and diverse model benchmarking.

Custom Dataset: 1,080 files across 36 classes, 864 files for training and 216 for validation.

Initially started with 5 WAV files per class. Using a custom script, each file was augmented with pitch shifting and volume adjustments, creating 5 additional versions per original file. This expansion helps in adapting models to specific voice samples.

Data Preprocessing

The dataset is initially loaded with a default audio sequence length, and further steps ensure that each waveform is standardized to a consistent length (16,000 samples, equivalent to 1 second of audio at 16kHz).

Padding/Trimming:
If the audio is shorter than 1 second (16,000 samples), it is padded; if it's longer, it is trimmed to fit the exact duration, ensuring uniformity across samples.

tf.squeeze()

The squeeze function was applied to remove singleton dimensions from the audio tensors. This adjustment ensures that each audio sample is represented in the required shape. The function was used during dataset mapping for both training and validation sets to streamline data for efficient model processing.

Spectrogram Transformation:

The audio is converted to a time-frequency representation using Short-Time Fourier Transform (STFT) via tf.signal.stft(). This generates a complex matrix representing the frequency content over time. The magnitude is extracted using tf.abs(), giving a 2D matrix of amplitude values over time. An extra dimension is then added using tf.newaxis to prepare the spectrogram for input into a convolutional neural network, resulting in a 4D tensor (batch_size, height, width, channels) where height and width are time and frequency, and channels are for CNN compatibility.

Model Summary

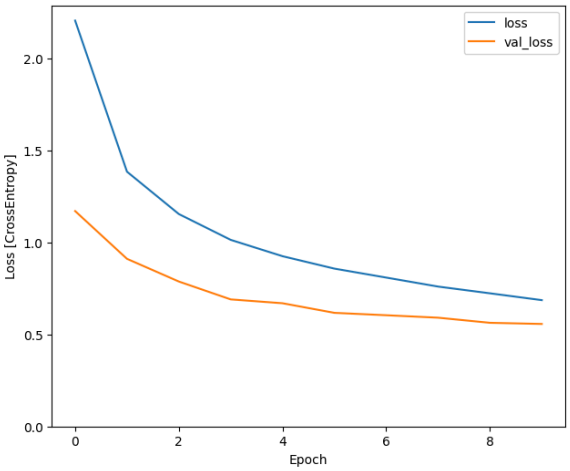
The chosen model captures time-frequency patterns in spectrograms while using fewer parameters than RNNs or transformers, making it computationally lighter.
Pooling layers make the model resilient to small variations in the input data.
It is easier to train and interpret compared to more complex models like transformers or RNNs.
The architecture is easy to expand or fine-tune for specific tasks.

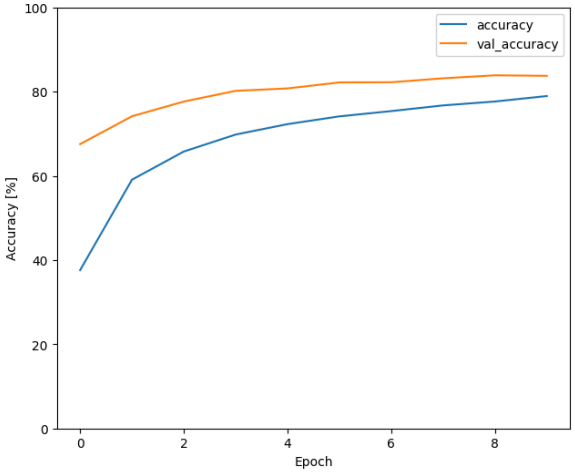
Input shape: (311, 129, 1)
Model: "sequential"

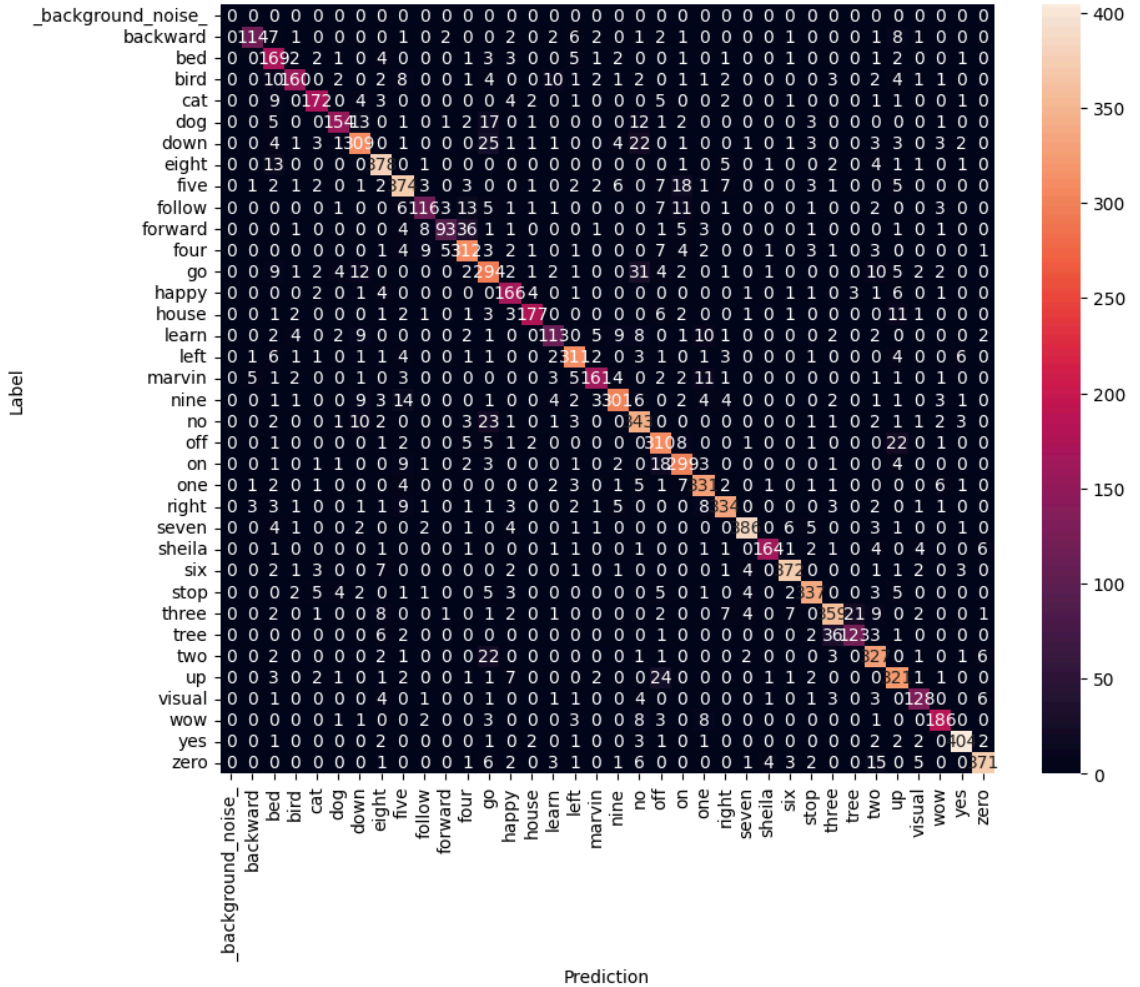
Layer (type)	Output Shape	Param #
resizing (Resizing)	(None, 32, 32, 1)	0
normalization (Normalization)	(None, 32, 32, 1)	3
conv2d (Conv2D)	(None, 30, 30, 32)	320
conv2d_1 (Conv2D)	(None, 28, 28, 64)	18,496
max_pooling2d (MaxPooling2D)	(None, 14, 14, 64)	0
dropout (Dropout)	(None, 14, 14, 64)	0
flatten (Flatten)	(None, 12544)	0
dense (Dense)	(None, 128)	1,605,760
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 33)	4,257

Total params: 1,628,836 (6.21 MB)
Trainable params: 1,628,833 (6.21 MB)
Non-trainable params: 3 (16.00 B)

Model Performance







Before FineTuning:

```
166/166 ━━━━━━━━━━━━━━━━━━━━ 25s 150ms/step - accuracy: 0.8436 - loss: 0.5392
{'accuracy': 0.8455736637115479, 'loss': 0.5361676812171936}
```

After FineTuning on own voice dataset:
At 16kHz SR:

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 120ms/step - accuracy: 0.8366 - loss: 0.5687
{'accuracy': 0.8309859037399292, 'loss': 0.5820963978767395}
```

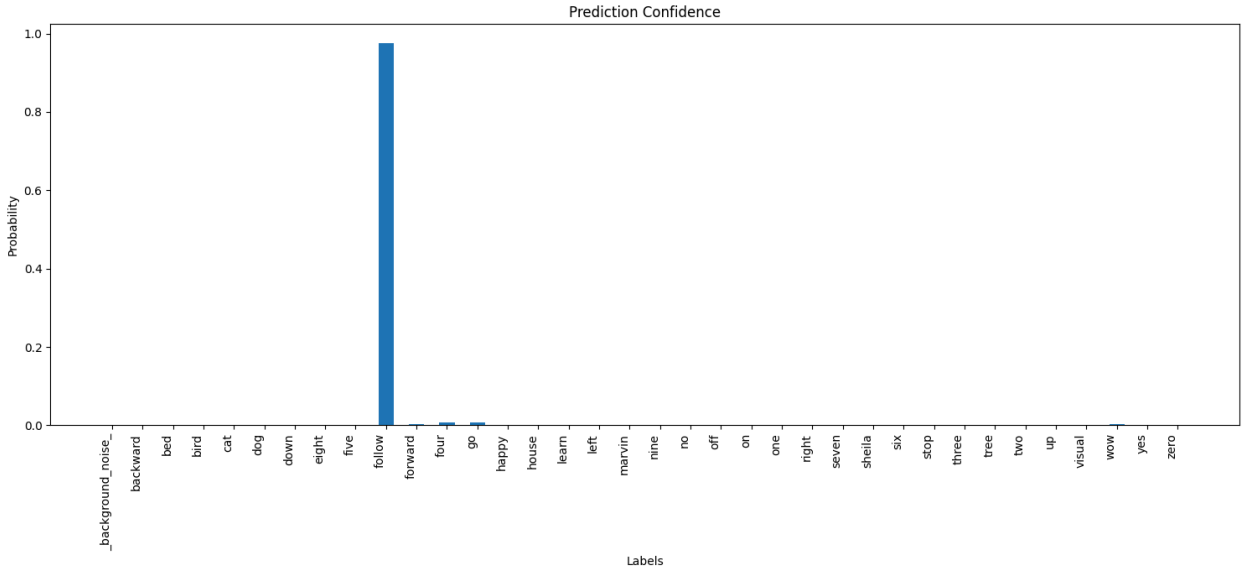
At 40kHz SR:

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 195ms/step - accuracy: 0.9962 - loss: 0.0501
{'accuracy': 0.9906103014945984, 'loss': 0.06075087562203407}
```

Top One Error

```
Top-One Error: 0.1408
```

Prediction



Prediction output for “follow” wav file

Code Summary

Libraries Used:

TensorFlow and TensorFlow Datasets: For data handling and model creation.
Librosa and Soundfile: For audio processing and augmentation.
Matplotlib and Seaborn: For data visualization.

Steps Followed:

1. **Data Loading and Preparation:**

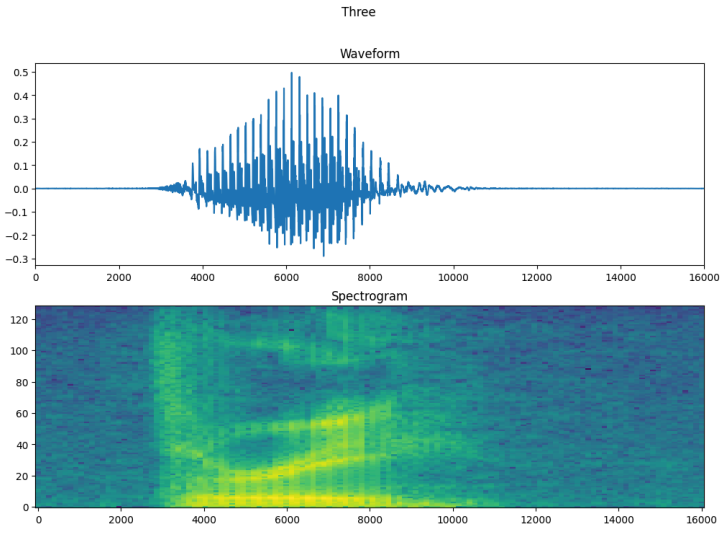
```
train_ds, val_ds = tf.keras.utils.audio_dataset_from_directory(
    directory=data_dir,
    batch_size=64,
    validation_split=0.2,
    seed=0,
    output_sequence_length=16000,
    subset='both')
label_names = np.array(train_ds.class_names)
print()
print("label names:", label_names)
```

```
Found 105835 files belonging to 36 classes.
Using 84668 files for training.
Using 21167 files for validation.

label names: ['_background_noise_' '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']
```

2. **Data Preprocessing:**

```
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)
```

3. **Spectrogram Visualization:**



4. **Model Creation:**

Convolutional Neural Network (CNN) with layers including resizing, normalization, convolutional, and dense layers.

Compiled the model with Adam optimizer and sparse categorical cross-entropy loss.

5. **Model Training:**

Trained the model on the spectrogram dataset with early stopping based on validation loss.

```
model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy'],
)
```

```
EPOCHS = 10
history = model.fit(
    train_spectrogram_ds,
    validation_data=val_spectrogram_ds,
    epochs=EPOCHS,
    callbacks=tf.keras.callbacks.EarlyStopping(verbose=1, patience=2),
)
```

6. **Evaluation and Prediction:**

```
def calculate_top_one_error(model, dataset):

  correct_predictions = 0
  total_predictions = 0

  for audio, labels in dataset:
    predictions = model.predict(audio)
    predicted_classes = tf.argmax(predictions, axis=1)
    correct_predictions += tf.reduce_sum(tf.cast(tf.equal(predicted_classes, tf.cast(labels, tf.int64)), tf.int32)).numpy()
    total_predictions += labels.shape[0]

  top_one_error = 1 - (correct_predictions / total_predictions)
  return top_one_error

top_one_error = calculate_top_one_error(model, test_spectrogram_ds)
print(f"Top-One Error: {top_one_error:.4f}")
```

7. **Custom Dataset:**

```
history_fine_tuned = model.fit(
    my_voice_spectrogram_ds,
    validation_data=my_voice_val_spectrogram_ds,
    epochs=15,
    callbacks=tf.keras.callbacks.EarlyStopping(verbose=1, patience=2)
)
```