RECOGNISE MY VOICE COMMANDS

SPEECH PROCESSING AND SYNTHESIS

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

Name: Atin Arora
Roll Number: 102103548

Submitted to:
Dr. Raghav B. Venkataramaiyer

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY,

(A DEEMED TO BE UNIVERSITY),
PATIALA, PUNJAB
INDIA

SUMMARY

The paper introduces the "Speech Commands Dataset," a collection of spoken utterances for training keyword detection systems. Since effective models are intended for low-resource devices, they should be able to understand instructions such as "Yes," "No," and "Stop." The dataset enhances the reliability and comparability of the model. Baseline models with an accuracy of 88.2% are included. More than 100,000 words from 2,618 speakers make up this collection.

I've changed this project so that it can identify my voice using 900 audio samples from my own dataset. It reaches a 91% maximum frequency.

DATASET SUMMARY

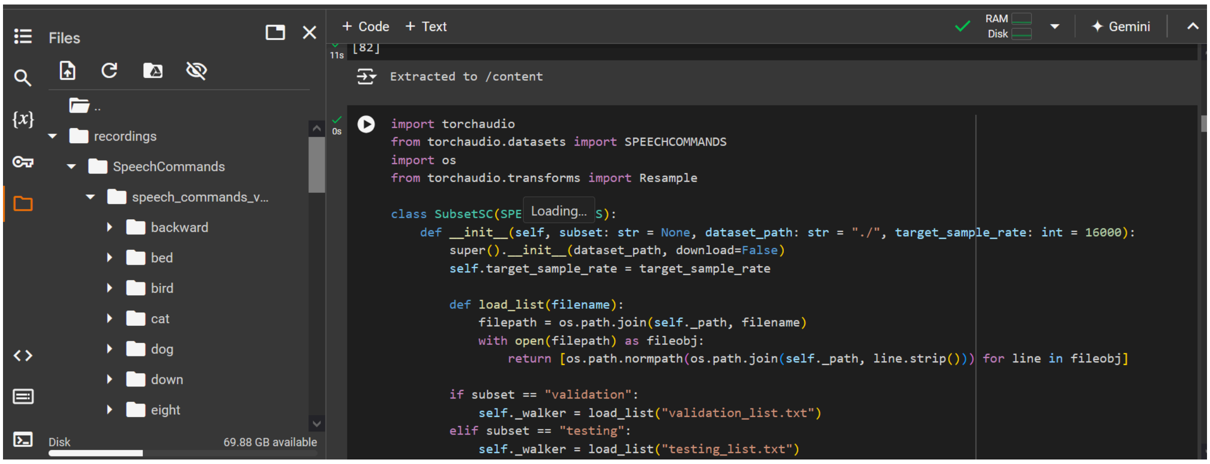
The study paper's Speech Commands Dataset includes over 105,829 audio recordings of 35 distinct words. Every audio file in the WAV format has a single spoken word that is sampled at 16 kHz. Since 2,618 speakers provided the dataset, a wide range of accents and pronunciations were guaranteed. With words like "Yes," "No," "Up," "Down," "Left," "Right," "On," "Off," "Stop," "Go," and other words like numerals (zero to nine), it primarily focuses on small-vocabulary keyword detecting tasks.

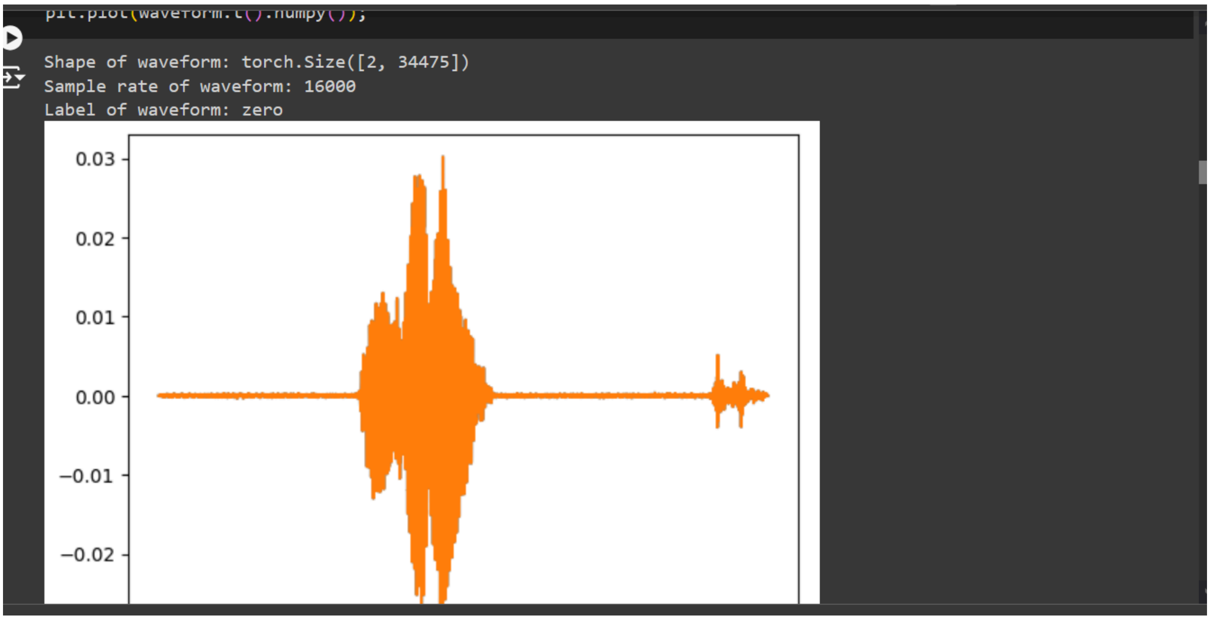
When uncompressed, the complete dataset is about 3.8 GB, or 2.7 GB when saved as a tar archive that has been compressed using gzip. It also comes with files including background noise to mimic actual situations and increase the robustness of the model.

PERFORMANCE MEASURES

The accuracy of the model came out to be maximum of 91% after 20 epochs.

CODE SNAPSHOTS





The 35 audio labels are commands that are said by users. The first few files are people saying “backward”.

```python
def train(model, epoch, log_interval):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):

        data = data.to(device)
        target = target.to(device)

        # apply transform and model on whole batch directly on device
        data = transform(data.contiguous()) # Ensure data is contiguous before applying transform
        output = model(data)

        # negative log-likelihood for a tensor of size (batch x 1 x n_output)
        loss = F.nll_loss(output.squeeze(), target)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # print training stats
        if batch_idx % log_interval == 0:
            print(f"Train Epoch: {epoch} [{batch_idx * len(data)}/{len(train_loader.dataset)} ({100. * batch_
```

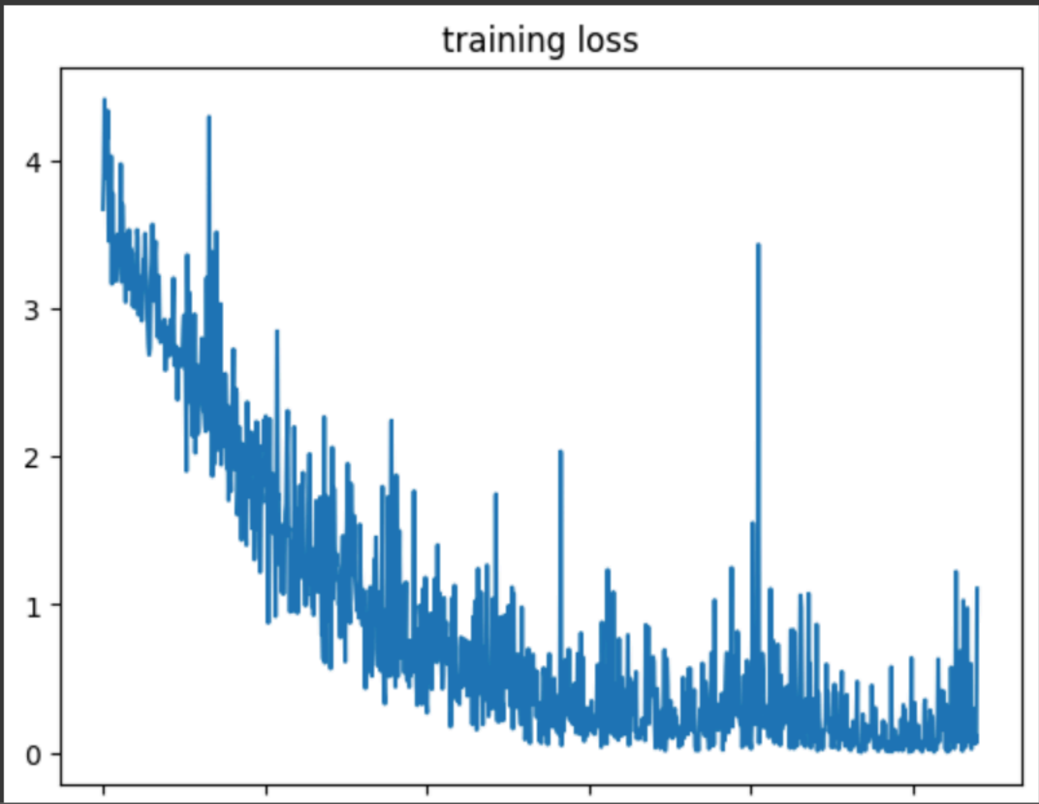
```python
log_interval = 5
n_epoch = 20

pbar_update = 1 / (len(train_loader) + len(test_loader))
losses = []

# The transform needs to live on the same device as the model and the data.
transform = transform.to(device)
with tqdm(total=n_epoch) as pbar:
    for epoch in range(1, n_epoch + 1):
        train(model, epoch, log_interval)
        test(model, epoch)
        scheduler.step()

# Let's plot the training loss versus the number of iteration.
plt.plot(losses);
plt.title("training loss");
```

```
Train Epoch: 18 [0/535 (0%)]     Loss: 0.391568
 86%|████████▌ | 17.10144927536183/20 [02:02<00:19,  6.74s/it] Train Ep
 86%|████████▌ | 17.188405797100955/20 [02:03<00:18,  6.55s/it]Train Ep
 86%|████████▌ | 17.246376811593706/20 [02:03<00:18,  6.69s/it]Train Ep
 87%|████████▋ | 17.33333333333283/20 [02:04<00:17,  6.47s/it] Train Ep
 87%|████████▋ | 17.39130434782558/20 [02:04<00:17,  6.62s/it] Train Ep
 87%|████████▋ | 17.449275362318332/20 [02:05<00:16,  6.40s/it]Train Ep
 88%|████████▊ | 17.536231884057457/20 [02:05<00:15,  6.44s/it]Train Ep
 88%|████████▊ | 17.623188405796583/20 [02:06<00:14,  6.08s/it]Train Ep
 88%|████████▊ | 17.681159420289333/20 [02:06<00:14,  6.23s/it]Train Ep
 89%|████████▉ | 17.76811594202846/20 [02:07<00:13,  6.25s/it] Train Ep
 90%|█████████ | 18.028985507245835/20 [02:08<00:10,  5.54s/it]
Test Epoch: 18  Accuracy: 132/145 (91%)
```




```
    fileformat = "wav"
    filename = f"_audio.{fileformat}"
    AudioSegment.from_file(BytesIO(b)).export(filename, format=fileformat)
    return torchaudio.load(filename)


# Detect whether notebook runs in google colab
if "google.colab" in sys.modules:
    waveform, sample_rate = record()
    print(f"Predicted: {predict(waveform)}.")
    ipd.Audio(waveform.numpy(), rate=sample_rate)
```

```
Recording started for 1 seconds.
Recording ended.
Predicted: no.
/usr/local/lib/python3.10/dist-packages/IPython/lib/display.py:174: RuntimeWarning: invalid value encountered :
  scaled = data / normalization_factor * 32767
/usr/local/lib/python3.10/dist-packages/IPython/lib/display.py:175: RuntimeWarning: invalid value encountered :
  return scaled.astype("<h").tobytes(), nchan
```