LAB EVALUATION REPORT FOR THE

SPEECH PROCESSING & SYNTHESIS (UCS749)

SUBMITTED BY

NALIN SINGHAL

102153021

PROJECT TITLE

RECOGNISE MY VOICE COMMANDS

COURSE COORDINATOR

RAGHAV B. VENKATARAMAIYER

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

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY, PATIALA

SESSION: JULY-DECEMBER, 2024

DATE OF SUBMISSION: 11/09/2024

SUMMARY

The paper brings into discussion the so called ‘’Speech Commands Dataset’’ which is a database of spoken words for keyword spotting training. Since it aims to employ energy-efficient devices, the efficient models should be able to recognise words such as “Yes,” “No,” and “Stop.” The benchmark models of the dataset are established at the efficacy of 88. 2 on the grounds that the figures facilitate enhancement of model quality and comparability as well as replication. This collection comprises than 100 000 words from 2628 speakers.

DATASET

Speech Commands Dataset being discussed in the research paper consists of more than 105,829 sample audio recordings with 35 spoken words. The recordings are in WAV, the sampling rate used is 16kHz and each of the audio file is of one word said verbally. To begin with, the set was compiled using the utterances of 2618 people in order to provide as diverse variants of pronunciation and intonations as possible. It is mostly used for micro-vocabulary fully keywords tasks these include; Yes, No, Up, Down, Left, Right, On, Off, Stop, Go and other words in form of digits, 0,1,2,3,4,5, 6,7, 8, 9.

All together the dataset represented here is about 3. in its uncompressed state it measures 8 GB or 2. , and, when uncompressed, will be about 7 GB if stored in gzip-compressed tar archive. More, it has background noise files to proximity with actual situations and enhance the model efficiency.

Key details:

Words: 35 target words and they are; digits, commands and auxiliary words.

Recordings: 105,829 utterances.

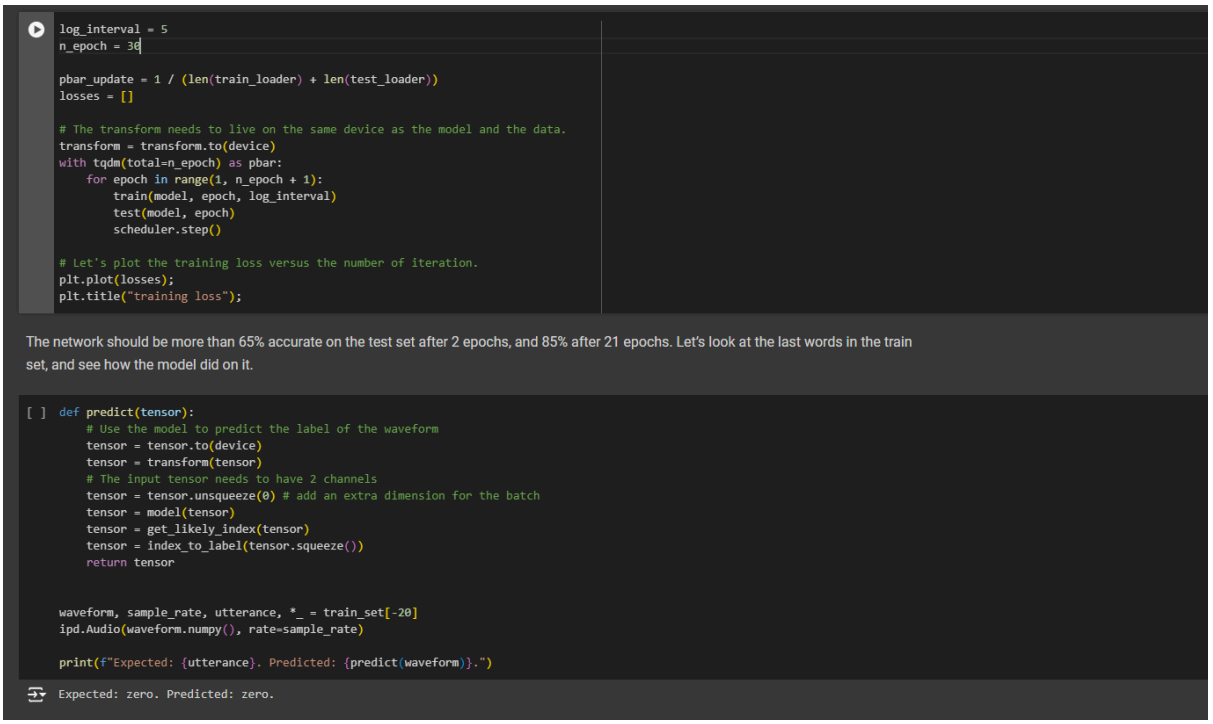
Speakers: 2,618 speakers where each was given a subsequent number.

File Format: WAV files at one second with 16-bit Pulse Code Modulation at a sample rate of 16000 Hz.

Size: ~3. 8 GB uncompressed, ~2. 7 GB compressed.

This dataset is intended for the development of on-Device keyword recognition models for conditions low resource conditions.

CODE SNAPSHOTS



```
log_interval = 5
n_epoch = 30

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");
```

The network should be more than 65% accurate on the test set after 2 epochs, and 85% after 21 epochs. Let's look at the last words in the train set, and see how the model did on it.

```
[ ] def predict(tensor):
        # Use the model to predict the label of the waveform
        tensor = tensor.to(device)
        tensor = transform(tensor)
        # The input tensor needs to have 2 channels
        tensor = tensor.unsqueeze(0) # add an extra dimension for the batch
        tensor = model(tensor)
        tensor = get_likely_index(tensor)
        tensor = index_to_label(tensor.squeeze())
        return tensor


    waveform, sample_rate, utterance, *_ = train_set[-20]
    ipd.Audio(waveform.numpy(), rate=sample_rate)

    print(f"Expected: {utterance}. Predicted: {predict(waveform)}.")
```

```
Expected: zero. Predicted: zero.
```

```python
def pad_sequence(batch):
    # Make all tensor in a batch the same length by padding with zeros
    batch = [item.t() for item in batch]
    batch = torch.nn.utils.rnn.pad_sequence(batch, batch_first=True, padding_value=0.)
    return batch.permute(0, 2, 1)


def collate_fn(batch):

    # A data tuple has the form:
    # waveform, sample_rate, label, speaker_id, utterance_number

    tensors, targets = [], []

    # Gather in lists, and encode labels as indices
    for waveform, _, label, *_ in batch:
        tensors += [waveform]
        targets += [label_to_index(label)]

    # Group the list of tensors into a batched tensor
    tensors = pad_sequence(tensors)
    targets = torch.stack(targets)

    return tensors, targets


batch_size = 15

if device == "cuda":
    num_workers = 256
    pin_memory = True
else:
    num_workers = 0
    pin_memory = False

train_loader = torch.utils.data.DataLoader(
    train_set,
    batch_size=batch_size,
    shuffle=True,
    collate_fn=collate_fn,
    num_workers=num_workers,
    pin_memory=pin_memory,
)
```

```
labels = sorted(list(set(datapoint[2] for datapoint in train_set)))
labels
```

```
['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']
```

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

```
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_idx / len(train_loader):.0f}%)]\tLoss: {loss.item():.6f}")

        # update progress bar
        pbar.update(pbar_update)
        # record loss
        losses.append(loss.item())
```