LAB EVAL- Conversational AI: Speech Processing and Synthesis

Jasmine Das-102117016-4CS1

Summary

The Speech Commands dataset is an open-source audio dataset designed for training and evaluating keyword spotting systems, providing a standardized way to compare models and encouraging collaboration and progress in the field. The primary goal of the Speech Commands dataset is to provide a way to build and test small models that detect when a single word is spoken, from a set of ten or fewer target words, with as few false positives as possible from background noise or unrelated speech.

Key findings

- The dataset provides a standardized way to compare keyword spotting models, which is essential for progress in the field.
- The dataset is designed to enable meaningful comparisons between different models' results in keyword spotting, focusing on distinguishing between audio-containing speech and clips containing none.
- The key finding is that the Version 2 dataset improves the accuracy of models, with a Top-One score of 88.2% on the training set and 89.7% on the Version 1 test set.
- There are some promising datasets to support general speech tasks, such as Mozilla's Common Voice, but they aren't adaptable to keyword spotting. This Speech Commands dataset aims to meet the special needs around building and testing on-device models, to enable model authors to demonstrate their architectures' accuracy using metrics comparable to other models and to give a simple way for teams to reproduce baseline models by training on identical data.
- The Speech Commands dataset is useful for training and evaluating a variety of models. The second version shows improved results on equivalent test data, compared to the original.

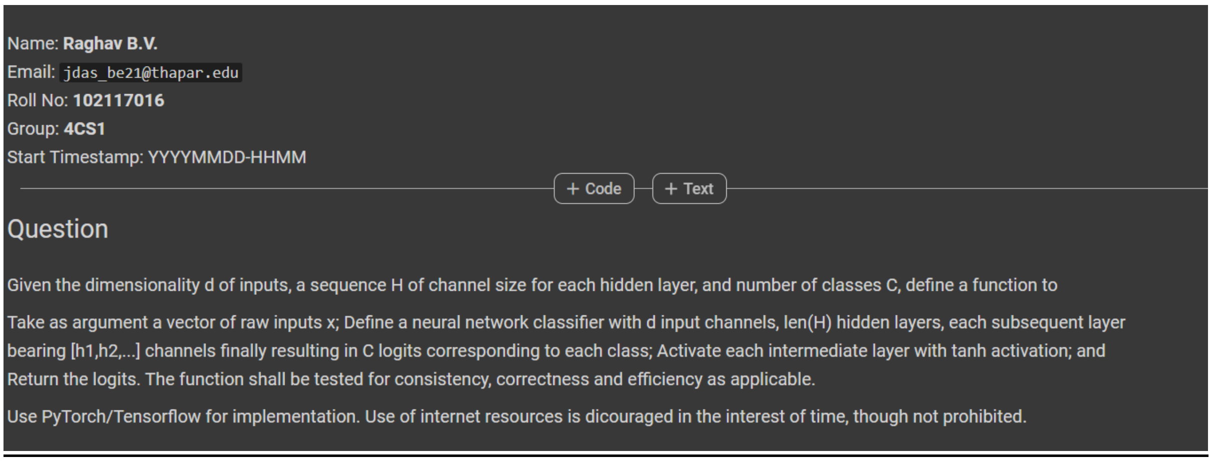
The dataset was collected using an open-source web-based application that recorded utterances using the Web Audio API. The application was designed to be quick and easy to use, to reduce the number of people who would fail to complete it. The dataset was created by recording audio clips of users speaking words from a list, with each clip stored as a one-second WAV file. The clips were then processed to remove quiet or silent recordings and to extract the loudest section of each clip. The methods used include training the default convolution model from the TensorFlow tutorial using the Version 1 and Version 2 training data and evaluating the models against the test sets.

The dataset provides a baseline for comparing the performance of different models in keyword spotting, focusing on distinguishing between audio-containing speech and clips containing none.

The conclusions drawn are that the Speech Commands dataset has shown to be useful for training and evaluating a variety of models. The second version shows improved results on equivalent test data, compared to the original.

DATASET:
https://drive.google.com/drive/folders/1iKWlCWFlia5rvHSz1Zbl4uf1hAZnC4F?usp=drive_link

SNIPPETS



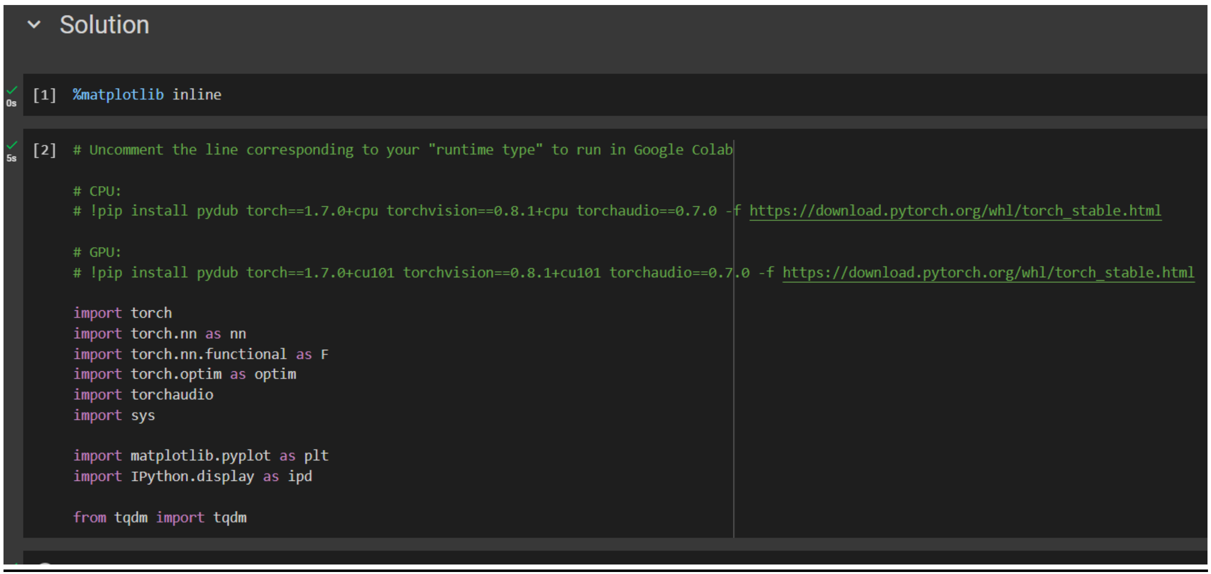
Name: **Raghav B.V.**
Email: `jdas_be21@thapar.edu`
Roll No: **102117016**
Group: **4CS1**
Start Timestamp: YYYYMMDD-HHMM

Question

Given the dimensionality d of inputs, a sequence H of channel size for each hidden layer, and number of classes C, define a function to

Take as argument a vector of raw inputs x; Define a neural network classifier with d input channels, len(H) hidden layers, each subsequent layer bearing [h1,h2,...] channels finally resulting in C logits corresponding to each class; Activate each intermediate layer with tanh activation; and Return the logits. The function shall be tested for consistency, correctness and efficiency as applicable.

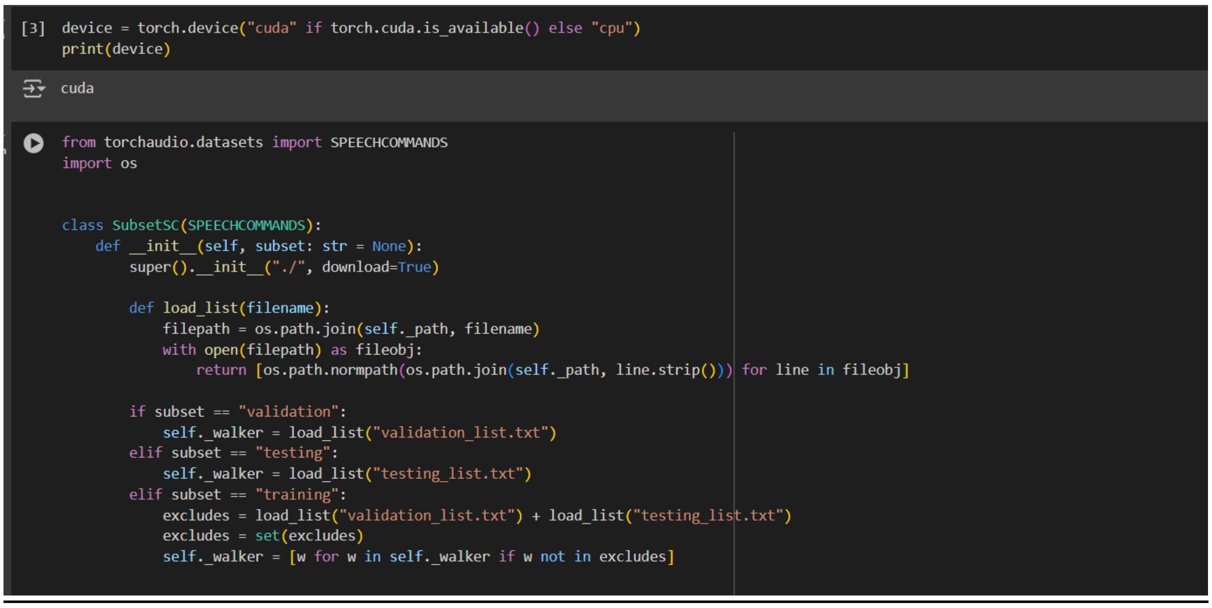
Use PyTorch/Tensorflow for implementation. Use of internet resources is dicouraged in the interest of time, though not prohibited.



Solution

```
[1] %matplotlib inline
```

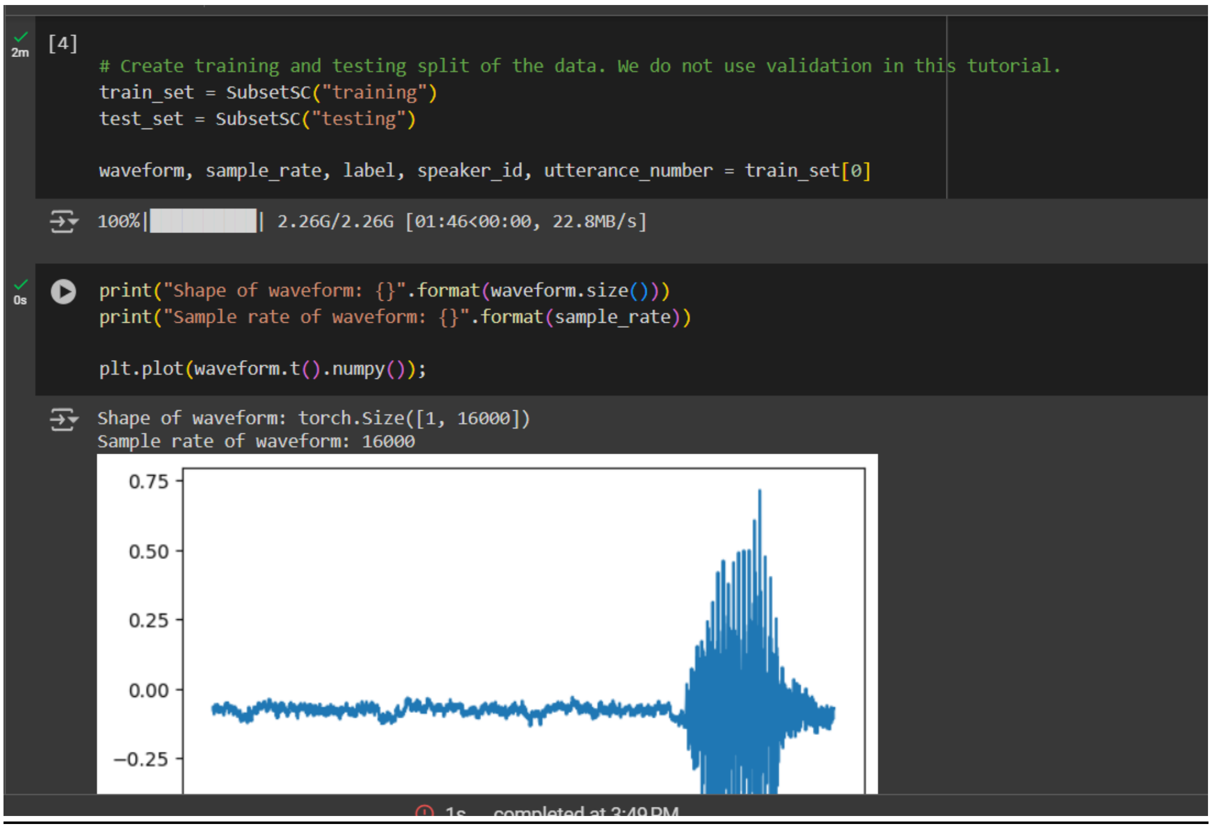
```
[2] # Uncomment the line corresponding to your "runtime type" to run in Google Colab

    # CPU:
    # !pip install pydub torch==1.7.0+cpu torchvision==0.8.1+cpu torchaudio==0.7.0 -f https://download.pytorch.org/whl/torch_stable.html

    # GPU:
    # !pip install pydub torch==1.7.0+cu101 torchvision==0.8.1+cu101 torchaudio==0.7.0 -f https://download.pytorch.org/whl/torch_stable.html

    import torch
    import torch.nn as nn
    import torch.nn.functional as F
    import torch.optim as optim
    import torchaudio
    import sys

    import matplotlib.pyplot as plt
    import IPython.display as ipd

    from tqdm import tqdm
```



```
[3] device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(device)
```

```
cuda
```

```
from torchaudio.datasets import SPEECHCOMMANDS
import os


class SubsetSC(SPEECHCOMMANDS):
    def __init__(self, subset: str = None):
        super().__init__("./", download=True)

        def load_list(filename):
            filepath = os.path.join(self._path, filename)
            with open(filepath) as fileobj:
                return [os.path.normpath(os.path.join(self._path, line.strip())) for line in fileobj]

        if subset == "validation":
            self._walker = load_list("validation_list.txt")
        elif subset == "testing":
            self._walker = load_list("testing_list.txt")
        elif subset == "training":
            excludes = load_list("validation_list.txt") + load_list("testing_list.txt")
            excludes = set(excludes)
            self._walker = [w for w in self._walker if w not in excludes]
```



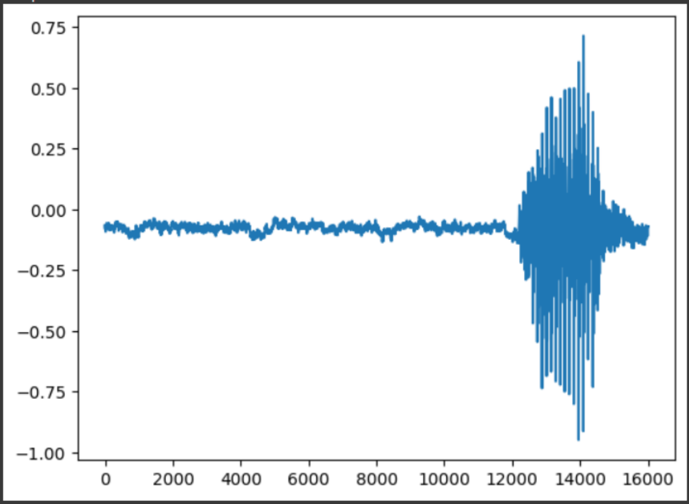
```
[4]
    # Create training and testing split of the data. We do not use validation in this tutorial.
    train_set = SubsetSC("training")
    test_set = SubsetSC("testing")

    waveform, sample_rate, label, speaker_id, utterance_number = train_set[0]
```

```
100%|██████████| 2.26G/2.26G [01:46<00:00, 22.8MB/s]
```

```
print("Shape of waveform: {}".format(waveform.size()))
print("Sample rate of waveform: {}".format(sample_rate))

plt.plot(waveform.t().numpy());
```

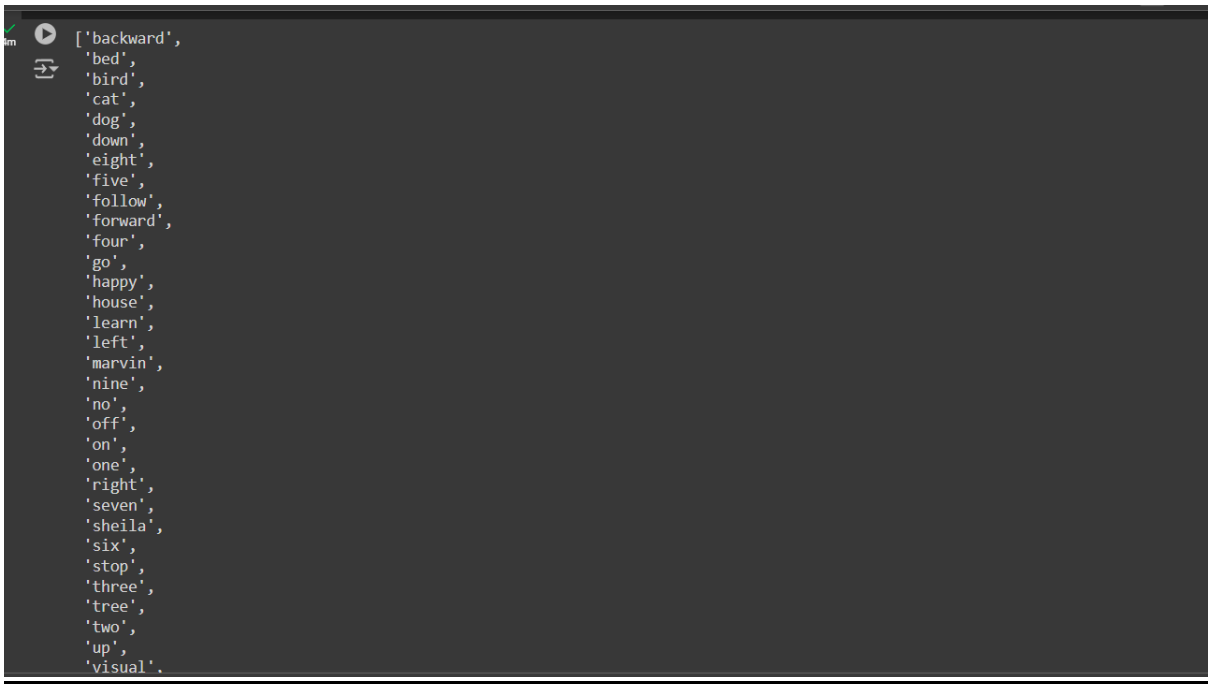
```
Shape of waveform: torch.Size([1, 16000])
Sample rate of waveform: 16000
```

```
Shape of waveform: torch.Size([1, 16000])
Sample rate of waveform: 16000
```

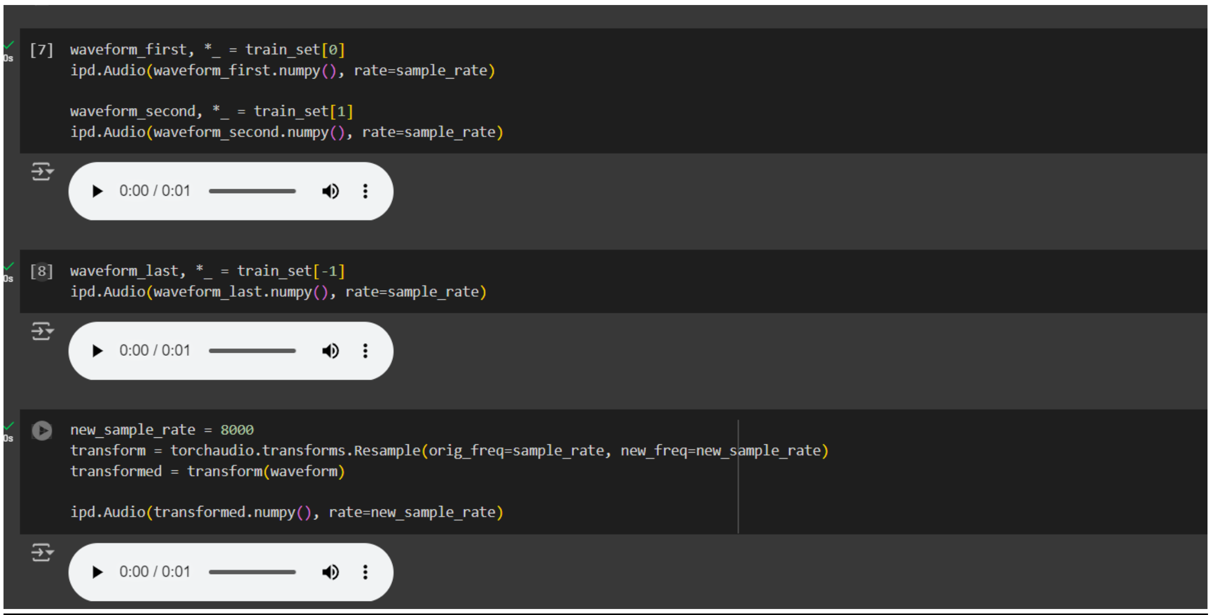


```
[6] labels = sorted(list(set(datapoint[2] for datapoint in train_set)))
    labels
```

```
['backward',
```



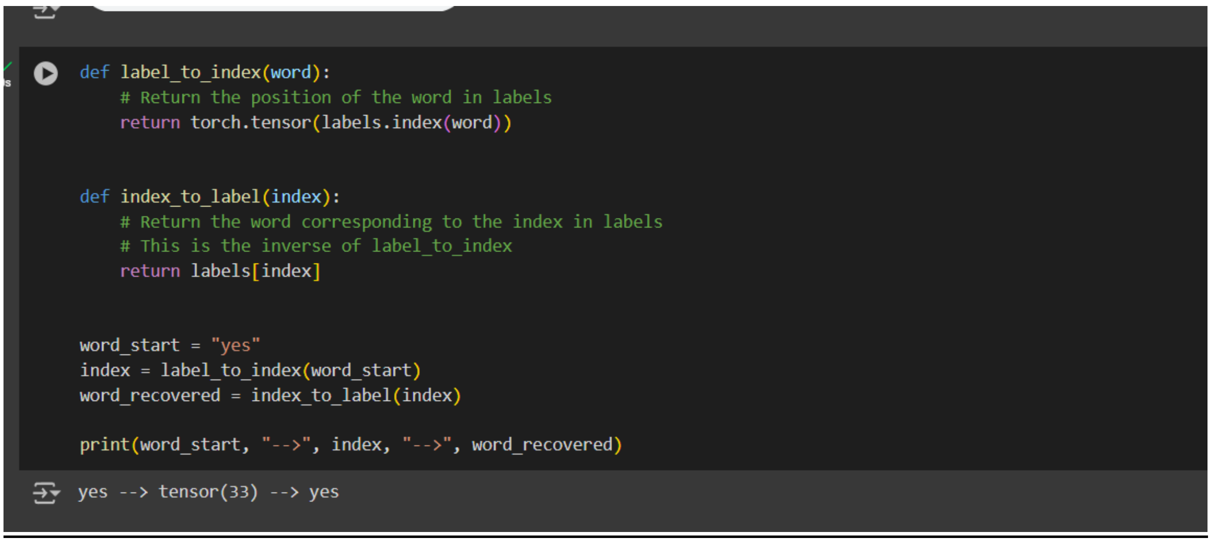
```
['backward',
 'bed',
 'bird',
 'cat',
 'dog',
 'down',
 'eight',
 'five',
 'follow',
 'forward',
 'four',
 'go',
 'happy',
 'house',
 'learn',
 'left',
 'marvin',
 'nine',
 'no',
 'off',
 'on',
 'one',
 'right',
 'seven',
 'sheila',
 'six',
 'stop',
 'three',
 'tree',
 'two',
 'up',
 'visual',
```



```
[7] waveform_first, *_ = train_set[0]
    ipd.Audio(waveform_first.numpy(), rate=sample_rate)

    waveform_second, *_ = train_set[1]
    ipd.Audio(waveform_second.numpy(), rate=sample_rate)
```

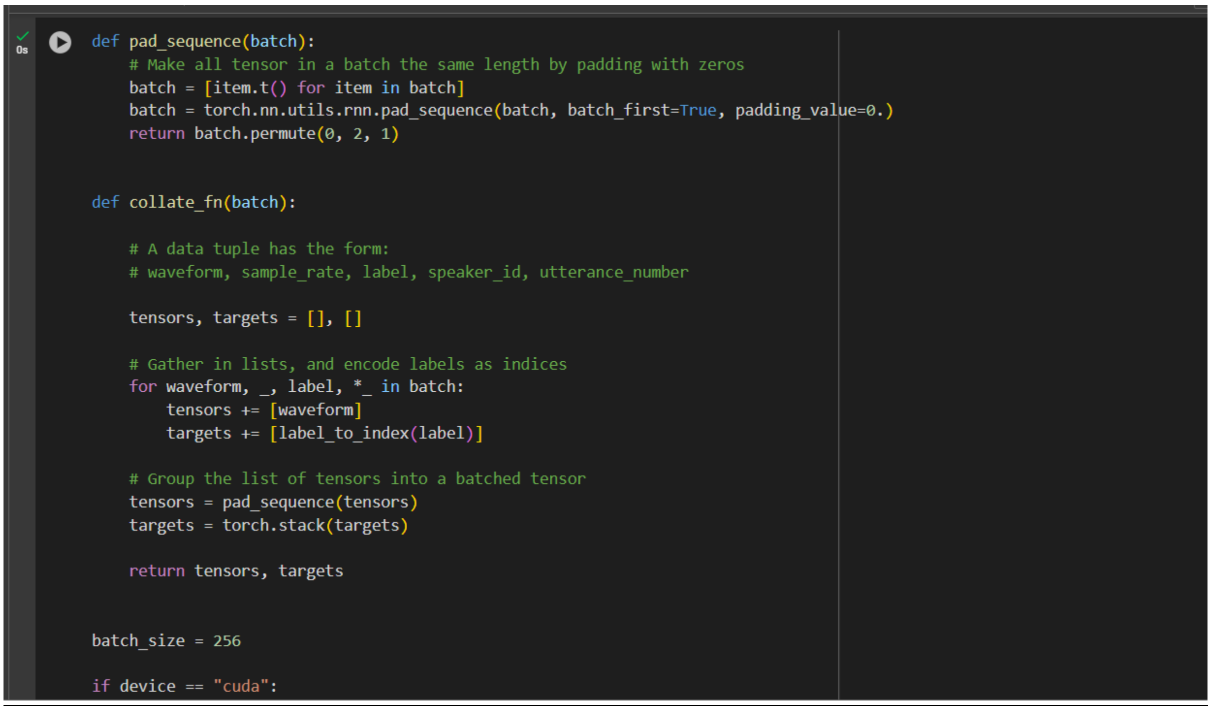
```
[8] waveform_last, *_ = train_set[-1]
    ipd.Audio(waveform_last.numpy(), rate=sample_rate)
```

```
new_sample_rate = 8000
transform = torchaudio.transforms.Resample(orig_freq=sample_rate, new_freq=new_sample_rate)
transformed = transform(waveform)

ipd.Audio(transformed.numpy(), rate=new_sample_rate)
```

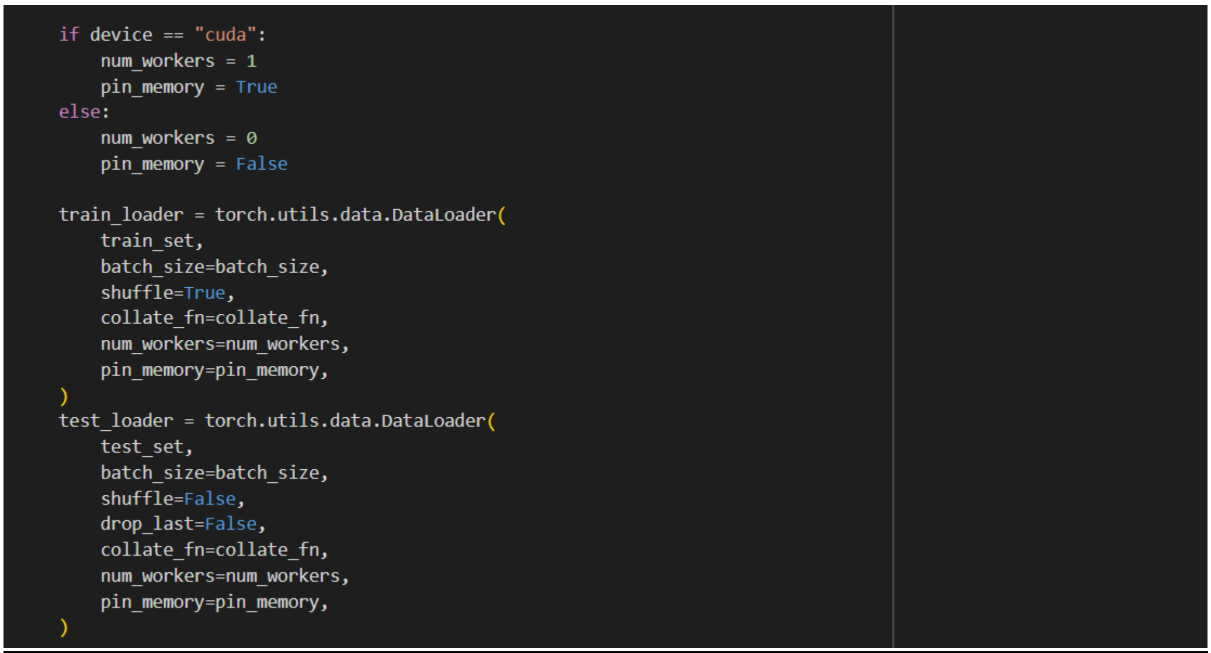


```
def label_to_index(word):
    # Return the position of the word in labels
    return torch.tensor(labels.index(word))


def index_to_label(index):
    # Return the word corresponding to the index in labels
    # This is the inverse of label_to_index
    return labels[index]


word_start = "yes"
index = label_to_index(word_start)
word_recovered = index_to_label(index)

print(word_start, "-->", index, "-->", word_recovered)
```

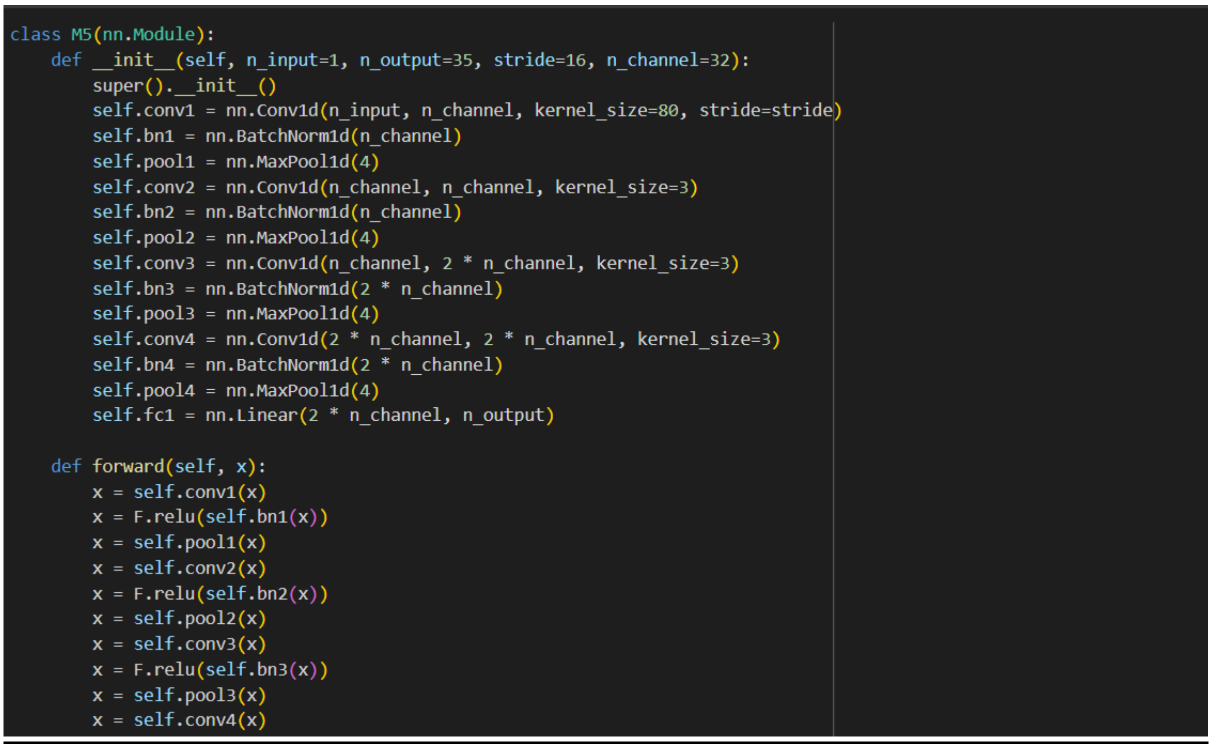
```
yes --> tensor(33) --> yes
```



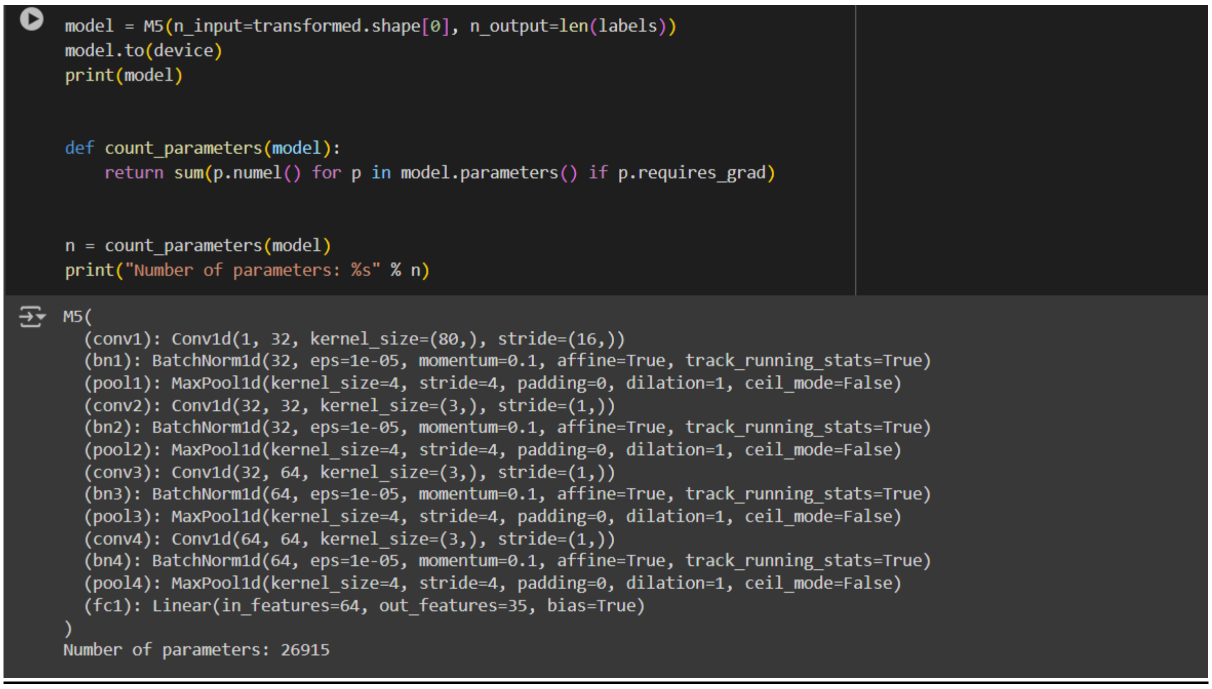
```
def pad_sequence(batch):
    # Make all tensor in a batch the same length by padding with zeros
    batch = [item.t() for item in batch]
    batch = torch.nn.utils.rnn.pad_sequence(batch, batch_first=True, padding_value=0.)
    return batch.permute(0, 2, 1)


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 = 256

if device == "cuda":
```



```
if device == "cuda":
    num_workers = 1
    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,
)
test_loader = torch.utils.data.DataLoader(
    test_set,
    batch_size=batch_size,
    shuffle=False,
    drop_last=False,
    collate_fn=collate_fn,
    num_workers=num_workers,
    pin_memory=pin_memory,
)
```

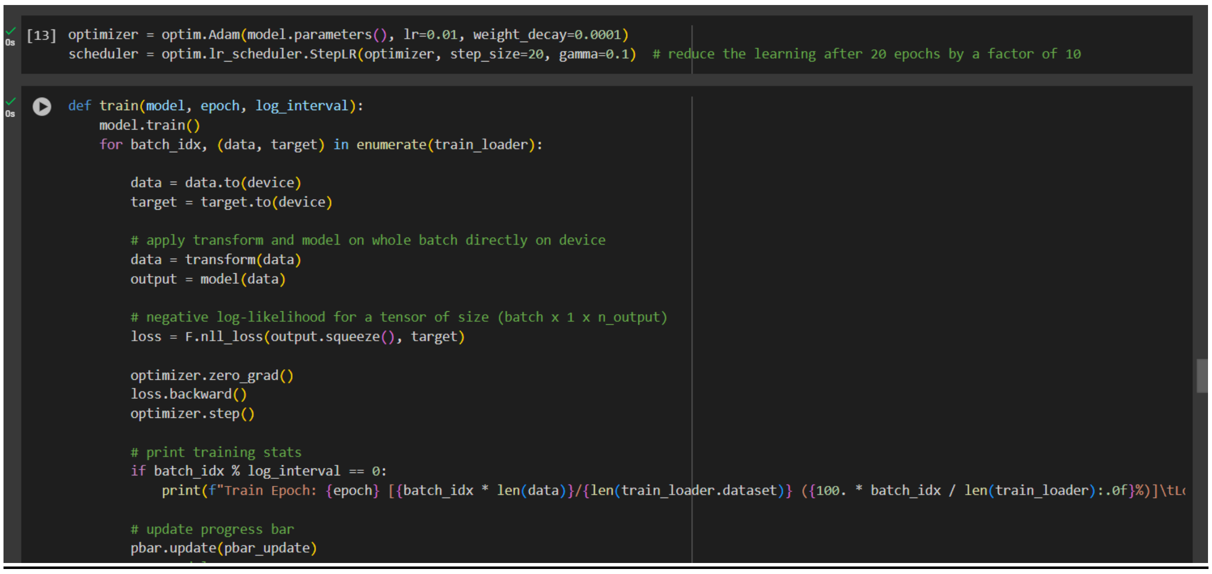


```python
class M5(nn.Module):
    def __init__(self, n_input=1, n_output=35, stride=16, n_channel=32):
        super().__init__()
        self.conv1 = nn.Conv1d(n_input, n_channel, kernel_size=80, stride=stride)
        self.bn1 = nn.BatchNorm1d(n_channel)
        self.pool1 = nn.MaxPool1d(4)
        self.conv2 = nn.Conv1d(n_channel, n_channel, kernel_size=3)
        self.bn2 = nn.BatchNorm1d(n_channel)
        self.pool2 = nn.MaxPool1d(4)
        self.conv3 = nn.Conv1d(n_channel, 2 * n_channel, kernel_size=3)
        self.bn3 = nn.BatchNorm1d(2 * n_channel)
        self.pool3 = nn.MaxPool1d(4)
        self.conv4 = nn.Conv1d(2 * n_channel, 2 * n_channel, kernel_size=3)
        self.bn4 = nn.BatchNorm1d(2 * n_channel)
        self.pool4 = nn.MaxPool1d(4)
        self.fc1 = nn.Linear(2 * n_channel, n_output)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(self.bn1(x))
        x = self.pool1(x)
        x = self.conv2(x)
        x = F.relu(self.bn2(x))
        x = self.pool2(x)
        x = self.conv3(x)
        x = F.relu(self.bn3(x))
        x = self.pool3(x)
        x = self.conv4(x)
```



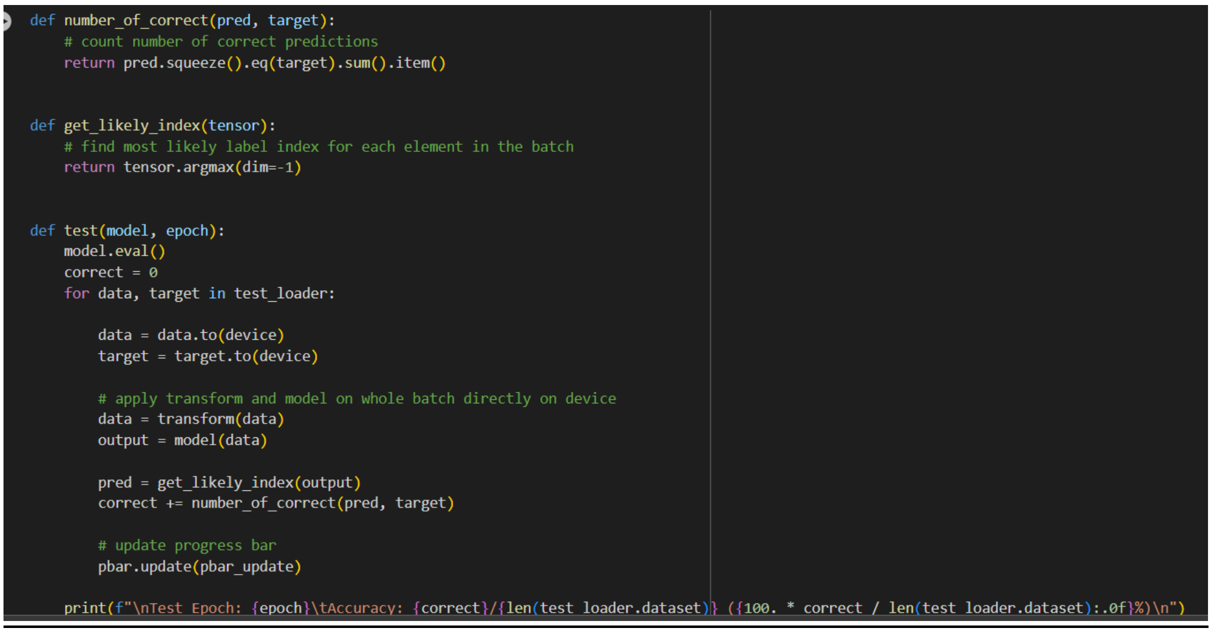
```python
model = M5(n_input=transformed.shape[0], n_output=len(labels))
model.to(device)
print(model)


def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)


n = count_parameters(model)
print("Number of parameters: %s" % n)
```

```
M5(
  (conv1): Conv1d(1, 32, kernel_size=(80,), stride=(16,))
  (bn1): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool1): MaxPool1d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv1d(32, 32, kernel_size=(3,), stride=(1,))
  (bn2): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool2): MaxPool1d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  (conv3): Conv1d(32, 64, kernel_size=(3,), stride=(1,))
  (bn3): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool3): MaxPool1d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  (conv4): Conv1d(64, 64, kernel_size=(3,), stride=(1,))
  (bn4): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool4): MaxPool1d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=64, out_features=35, bias=True)
)
Number of parameters: 26915
```

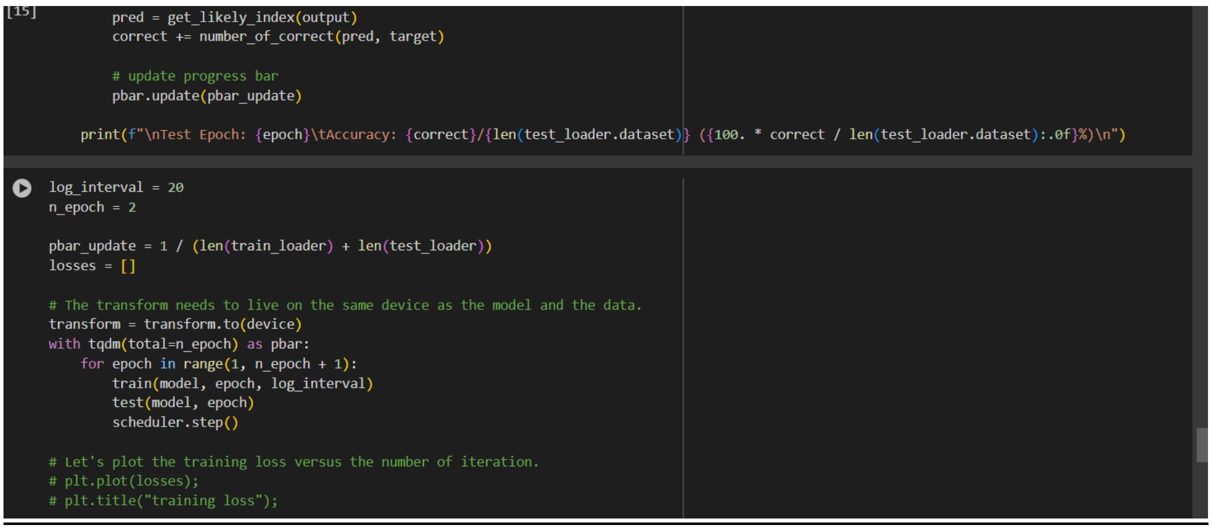


```
[13] optimizer = optim.Adam(model.parameters(), lr=0.01, weight_decay=0.0001)
     scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=20, gamma=0.1)  # reduce the learning after 20 epochs by a factor of 10
```

```
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)
        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}%)]\tL

        # update progress bar
        pbar.update(pbar_update)
```

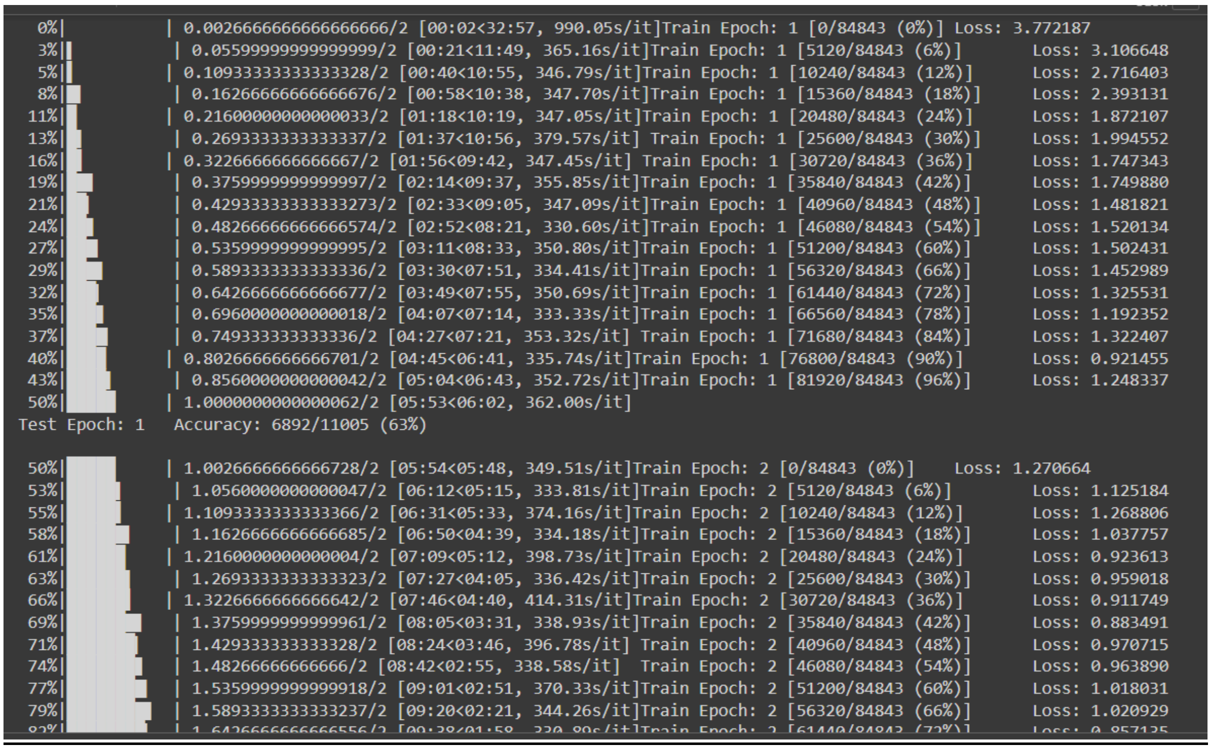


```
def number_of_correct(pred, target):
    # count number of correct predictions
    return pred.squeeze().eq(target).sum().item()


def get_likely_index(tensor):
    # find most likely label index for each element in the batch
    return tensor.argmax(dim=-1)


def test(model, epoch):
    model.eval()
    correct = 0
    for data, target in test_loader:

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

        # apply transform and model on whole batch directly on device
        data = transform(data)
        output = model(data)

        pred = get_likely_index(output)
        correct += number_of_correct(pred, target)

        # update progress bar
        pbar.update(pbar_update)

    print(f"\nTest Epoch: {epoch}\tAccuracy: {correct}/{len(test_loader.dataset)} ({100. * correct / len(test_loader.dataset):.0f}%)\n")
```

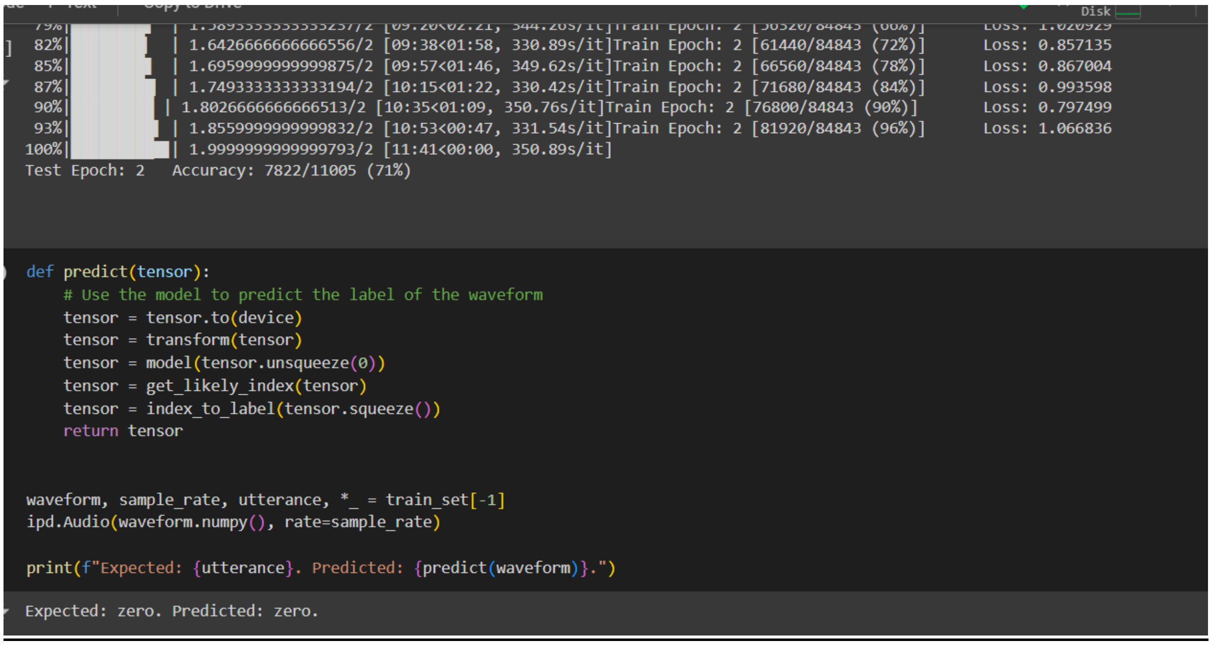


```
[15]        pred = get_likely_index(output)
            correct += number_of_correct(pred, target)

            # update progress bar
            pbar.update(pbar_update)

        print(f"\nTest Epoch: {epoch}\tAccuracy: {correct}/{len(test_loader.dataset)} ({100. * correct / len(test_loader.dataset):.0f}%)\n")
```

```
log_interval = 20
n_epoch = 2

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");
```



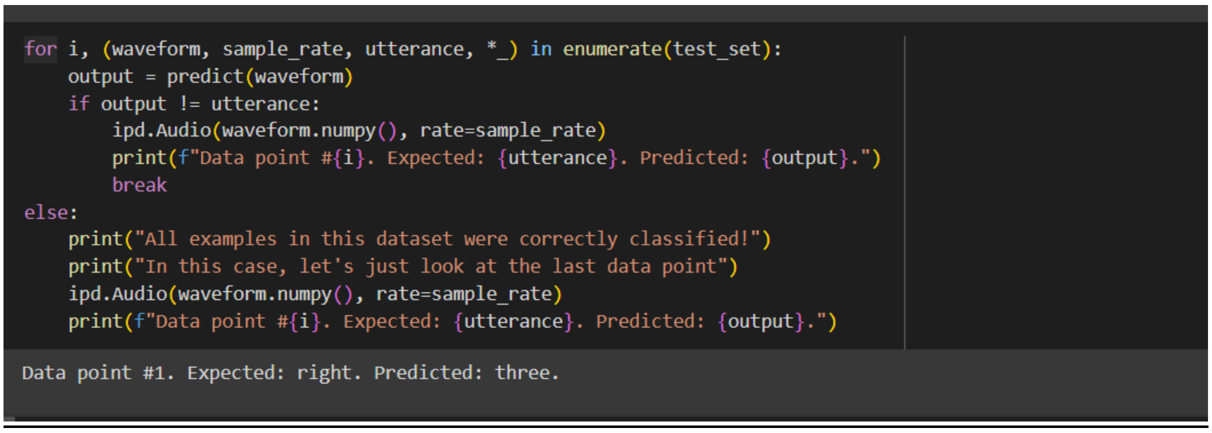
```
  0%|          | 0.0026666666666666666/2 [00:02<32:57, 990.05s/it]Train Epoch: 1 [0/84843 (0%)] Loss: 3.772187
  3%|▏          | 0.05599999999999999/2 [00:21<11:49, 365.16s/it]Train Epoch: 1 [5120/84843 (6%)]       Loss: 3.106648
  5%|▏         | 0.10933333333333328/2 [00:40<10:55, 346.79s/it]Train Epoch: 1 [10240/84843 (12%)]      Loss: 2.716403
  8%|▊          | 0.16266666666666676/2 [00:58<10:38, 347.70s/it]Train Epoch: 1 [15360/84843 (18%)]      Loss: 2.393131
 11%|█         | 0.21600000000000033/2 [01:18<10:19, 347.05s/it]Train Epoch: 1 [20480/84843 (24%)]      Loss: 1.872107
 13%|█          | 0.2693333333333337/2 [01:37<10:56, 379.57s/it] Train Epoch: 1 [25600/84843 (30%)]      Loss: 1.994552
 16%|█         | 0.3226666666666667/2 [01:56<09:42, 347.45s/it] Train Epoch: 1 [30720/84843 (36%)]      Loss: 1.747343
 19%|█▉         | 0.3759999999999997/2 [02:14<09:37, 355.85s/it]Train Epoch: 1 [35840/84843 (42%)]      Loss: 1.749880
 21%|██         | 0.42933333333333273/2 [02:33<09:05, 347.09s/it]Train Epoch: 1 [40960/84843 (48%)]      Loss: 1.481821
 24%|██▍        | 0.48266666666666574/2 [02:52<08:21, 330.60s/it]Train Epoch: 1 [46080/84843 (54%)]      Loss: 1.520134
 27%|██▋        | 0.5359999999999995/2 [03:11<08:33, 350.80s/it]Train Epoch: 1 [51200/84843 (60%)]      Loss: 1.502431
 29%|██▉        | 0.5893333333333336/2 [03:30<07:51, 334.41s/it]Train Epoch: 1 [56320/84843 (66%)]      Loss: 1.452989
 32%|███▏       | 0.6426666666666677/2 [03:49<07:55, 350.69s/it]Train Epoch: 1 [61440/84843 (72%)]      Loss: 1.325531
 35%|███▍       | 0.6960000000000018/2 [04:07<07:14, 333.33s/it]Train Epoch: 1 [66560/84843 (78%)]      Loss: 1.192352
 37%|███▋       | 0.749333333333336/2 [04:27<07:21, 353.32s/it] Train Epoch: 1 [71680/84843 (84%)]      Loss: 1.322407
 40%|████      | 0.8026666666666701/2 [04:45<06:41, 335.74s/it]Train Epoch: 1 [76800/84843 (90%)]      Loss: 0.921455
 43%|████▎      | 0.8560000000000042/2 [05:04<06:43, 352.72s/it]Train Epoch: 1 [81920/84843 (96%)]      Loss: 1.248337
 50%|█████     | 1.0000000000000062/2 [05:53<06:02, 362.00s/it]
Test Epoch: 1   Accuracy: 6892/11005 (63%)

 50%|█████     | 1.0026666666666728/2 [05:54<05:48, 349.51s/it]Train Epoch: 2 [0/84843 (0%)]    Loss: 1.270664
 53%|█████▎     | 1.0560000000000047/2 [06:12<05:15, 333.81s/it]Train Epoch: 2 [5120/84843 (6%)]       Loss: 1.125184
 55%|█████▍    | 1.1093333333333366/2 [06:31<05:33, 374.16s/it]Train Epoch: 2 [10240/84843 (12%)]      Loss: 1.268806
 58%|█████▊     | 1.1626666666666685/2 [06:50<04:39, 334.18s/it]Train Epoch: 2 [15360/84843 (18%)]      Loss: 1.037757
 61%|██████    | 1.2160000000000004/2 [07:09<05:12, 398.73s/it]Train Epoch: 2 [20480/84843 (24%)]      Loss: 0.923613
 63%|██████▎    | 1.2693333333333323/2 [07:27<04:05, 336.42s/it]Train Epoch: 2 [25600/84843 (30%)]      Loss: 0.959018
 66%|██████▌   | 1.3226666666666642/2 [07:46<04:40, 414.31s/it]Train Epoch: 2 [30720/84843 (36%)]      Loss: 0.911749
 69%|██████▉    | 1.3759999999999961/2 [08:05<03:31, 338.93s/it]Train Epoch: 2 [35840/84843 (42%)]      Loss: 0.883491
 71%|███████    | 1.429333333333328/2 [08:24<03:46, 396.78s/it] Train Epoch: 2 [40960/84843 (48%)]      Loss: 0.970715
 74%|███████▍   | 1.48266666666666/2 [08:42<02:55, 338.58s/it]  Train Epoch: 2 [46080/84843 (54%)]      Loss: 0.963890
 77%|███████▋   | 1.5359999999999918/2 [09:01<02:51, 370.33s/it]Train Epoch: 2 [51200/84843 (60%)]      Loss: 1.018031
 79%|███████▉   | 1.5893333333333237/2 [09:20<02:21, 344.26s/it]Train Epoch: 2 [56320/84843 (66%)]      Loss: 1.020929
```



```
 82%|████████▏  | 1.6426666666666556/2 [09:38<01:58, 330.89s/it]Train Epoch: 2 [61440/84843 (72%)]      Loss: 0.857135
 85%|████████▍  | 1.6959999999999875/2 [09:57<01:46, 349.62s/it]Train Epoch: 2 [66560/84843 (78%)]      Loss: 0.867004
 87%|████████▋  | 1.7493333333333194/2 [10:15<01:22, 330.42s/it]Train Epoch: 2 [71680/84843 (84%)]      Loss: 0.993598
 90%|█████████ | 1.8026666666666513/2 [10:35<01:09, 350.76s/it]Train Epoch: 2 [76800/84843 (90%)]      Loss: 0.797499
 93%|█████████▎ | 1.8559999999999832/2 [10:53<00:47, 331.54s/it]Train Epoch: 2 [81920/84843 (96%)]      Loss: 1.066836
100%|██████████| 1.9999999999999793/2 [11:41<00:00, 350.89s/it]
Test Epoch: 2   Accuracy: 7822/11005 (71%)
```

```python
def predict(tensor):
    # Use the model to predict the label of the waveform
    tensor = tensor.to(device)
    tensor = transform(tensor)
    tensor = model(tensor.unsqueeze(0))
    tensor = get_likely_index(tensor)
    tensor = index_to_label(tensor.squeeze())
    return tensor


waveform, sample_rate, utterance, *_ = train_set[-1]
ipd.Audio(waveform.numpy(), rate=sample_rate)

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

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



```python
for i, (waveform, sample_rate, utterance, *_) in enumerate(test_set):
    output = predict(waveform)
    if output != utterance:
        ipd.Audio(waveform.numpy(), rate=sample_rate)
        print(f"Data point #{i}. Expected: {utterance}. Predicted: {output}.")
        break
else:
    print("All examples in this dataset were correctly classified!")
    print("In this case, let's just look at the last data point")
    ipd.Audio(waveform.numpy(), rate=sample_rate)
    print(f"Data point #{i}. Expected: {utterance}. Predicted: {output}.")
```

```
Data point #1. Expected: right. Predicted: three.
```

CODE LINK:

https://colab.research.google.com/github/JasmineDas5/102117016_SESS_LE1/blob/main/about-lab-eval/102117016_JasmineDas.ipynb