SPEECH LAB EVAL

NAME : PRIYANSHI KHARBANDA

ROLL NO. : 102103395

SUBGROUP : 4CO14

Paper Summary :

The paper **"Speech Commands: A Dataset for Fine-Grained Speech Recognition"** presents a dataset with 65,000 one-second recordings of 12 spoken commands, such as "yes," "no," "up," and "down." It aims to advance research in speech recognition by offering a standardized benchmark for fine-grained audio classification, focusing on real-world performance in command recognition.

Google Collab files link :

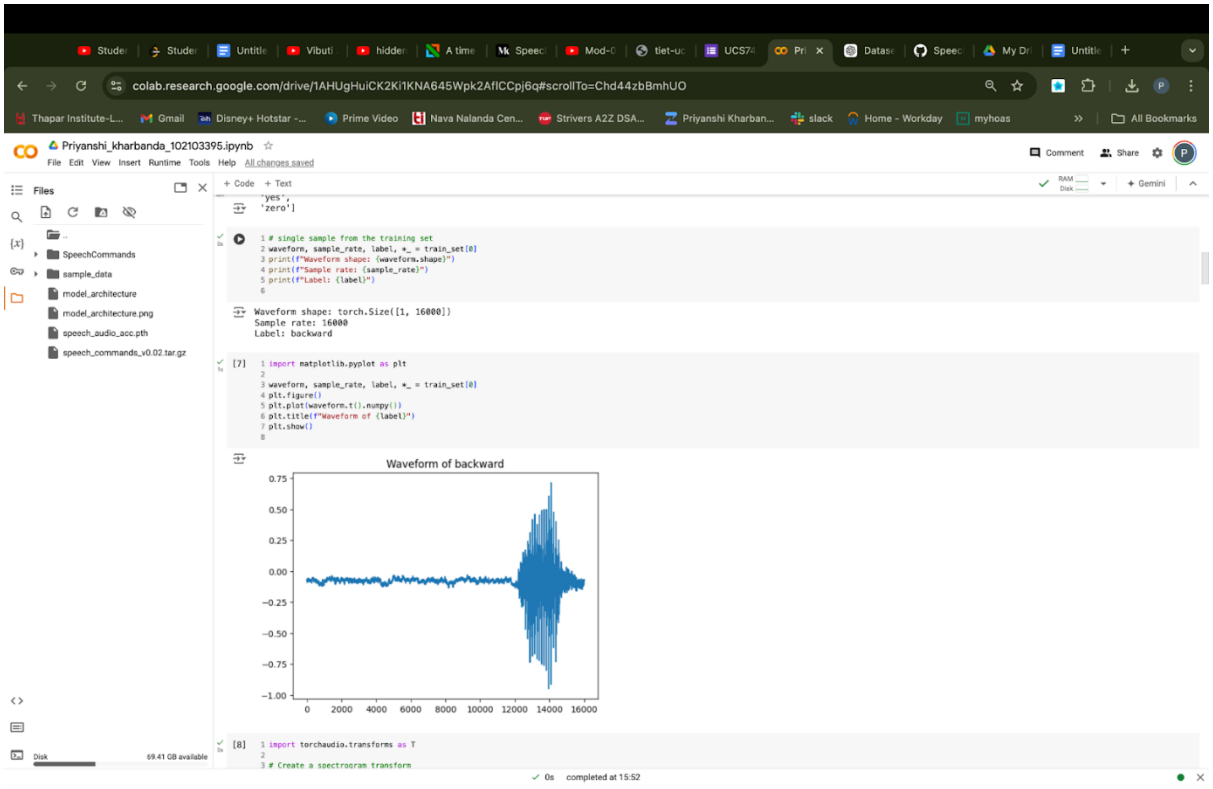
https://colab.research.google.com/drive/1dcbddNXmg2VLQ_4t2HFQZc86bKP_z6cC

https://colab.research.google.com/drive/1k2v-4TAlplImoT-EbDsSzEr4eQBCU640#scrollTo=SdDDteRj07ft

https://colab.research.google.com/drive/1auuc3uhHo1DOKkCUVxBDmp3nud7o6dVU?usp=sharing

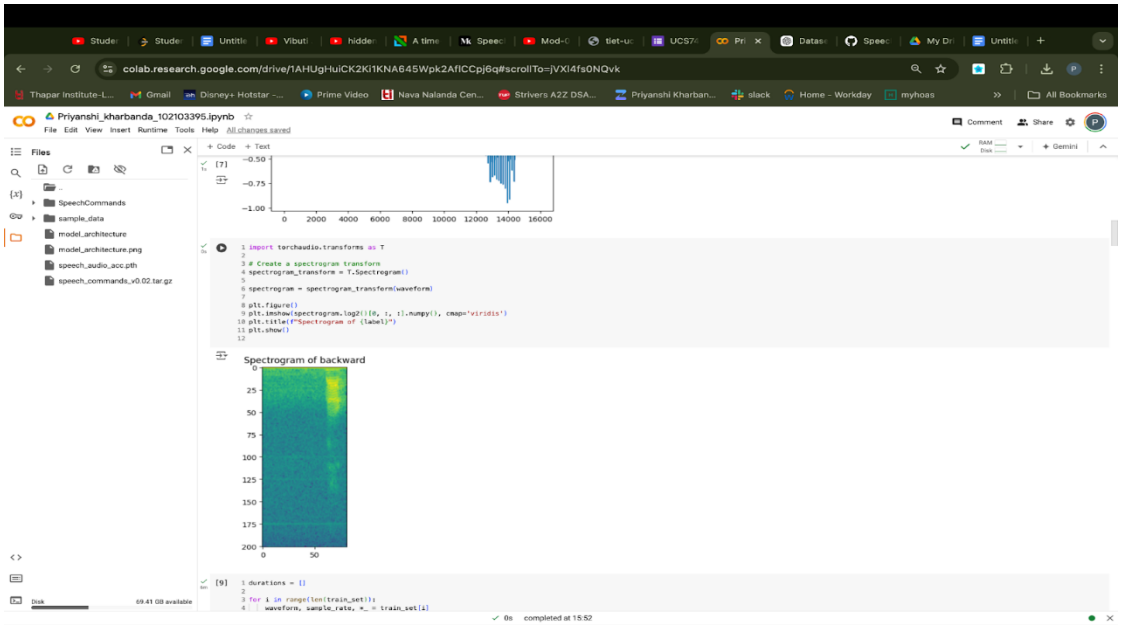
Analysis of data

1. A data point in the SPEECHCOMMANDS dataset is a tuple made of a waveform (the audio signal), the sample rate, the utterance (label), the ID of the speaker, the number of the utterance.



Number of training samples: 84843
Number of testing samples: 11005

2. After applying fourier transformation, here is what a spectogram of label backward looks like



3. Analysis done of duration of samples in our speechcommand dataset.

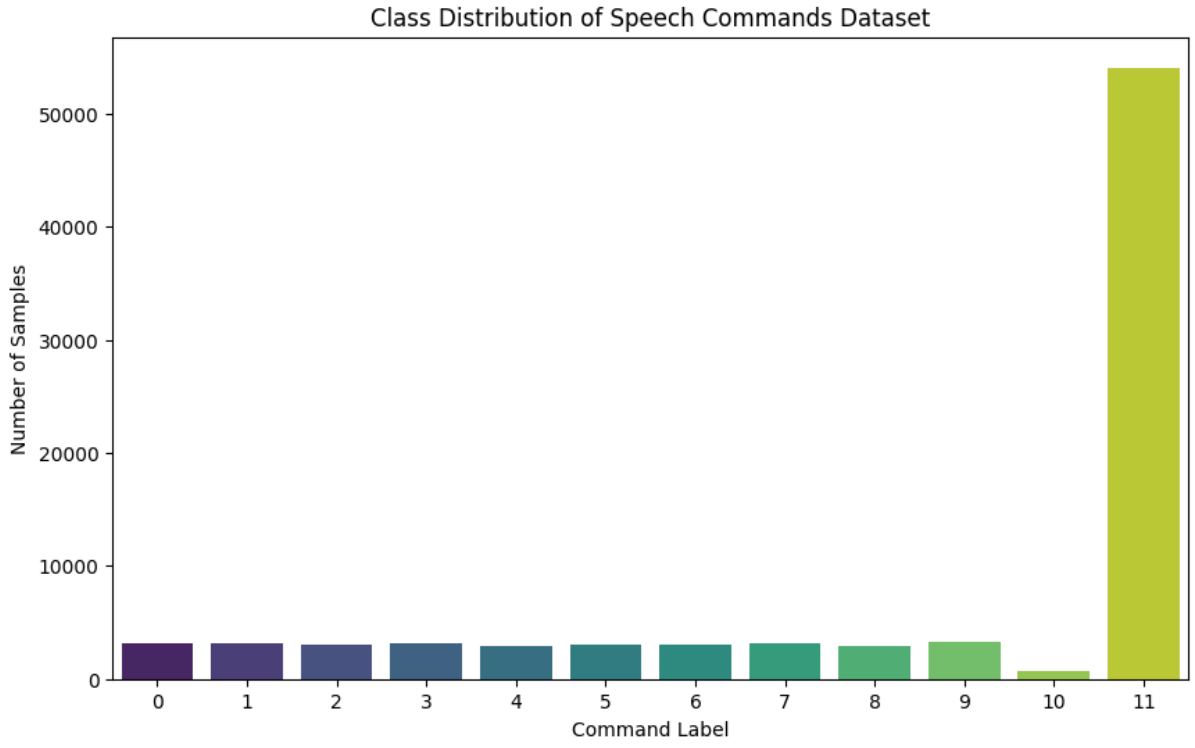
```python
durations = []

for i in range(len(train_set)):
    waveform, sample_rate, *_ = train_set[i]
    durations.append(waveform.shape[1] / sample_rate)

print(f"Average duration: {sum(durations) / len(durations)} seconds")
print(f"Maximum duration: {max(durations)} seconds")
print(f"Minimum duration: {min(durations)} seconds")
```

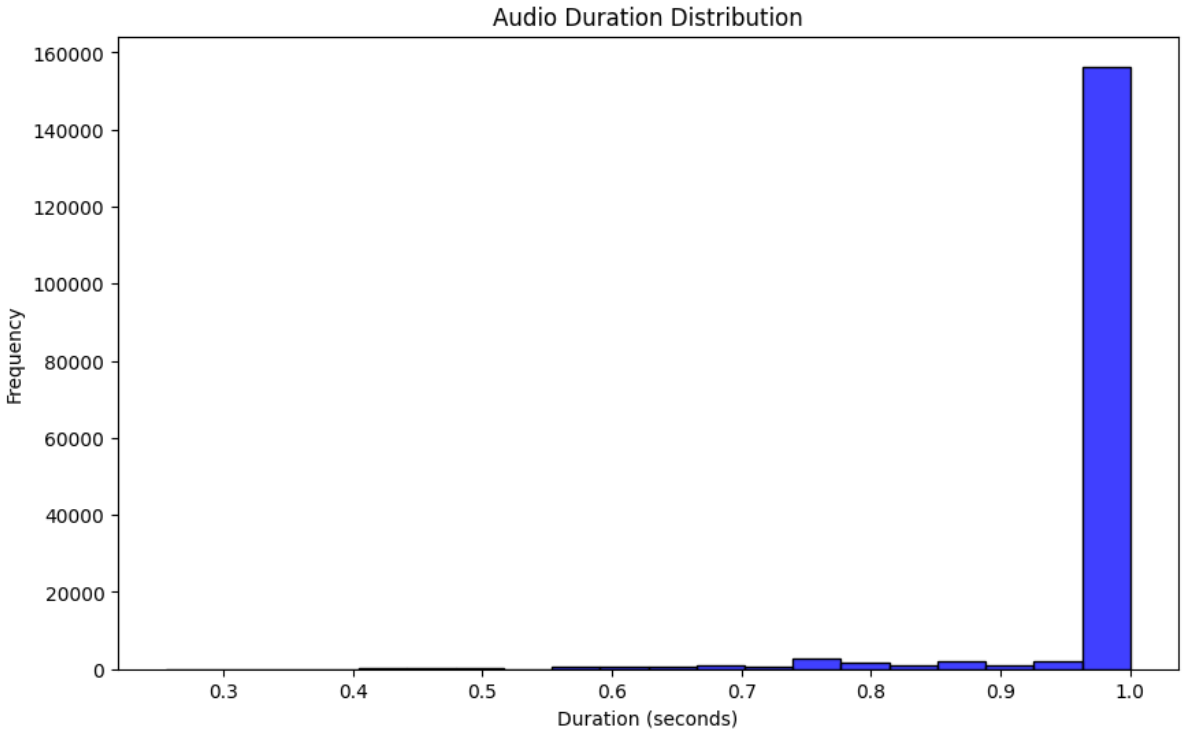
```
Average duration: 0.9807155429734978 seconds
Maximum duration: 1.0 seconds
Minimum duration: 0.256 seconds
```

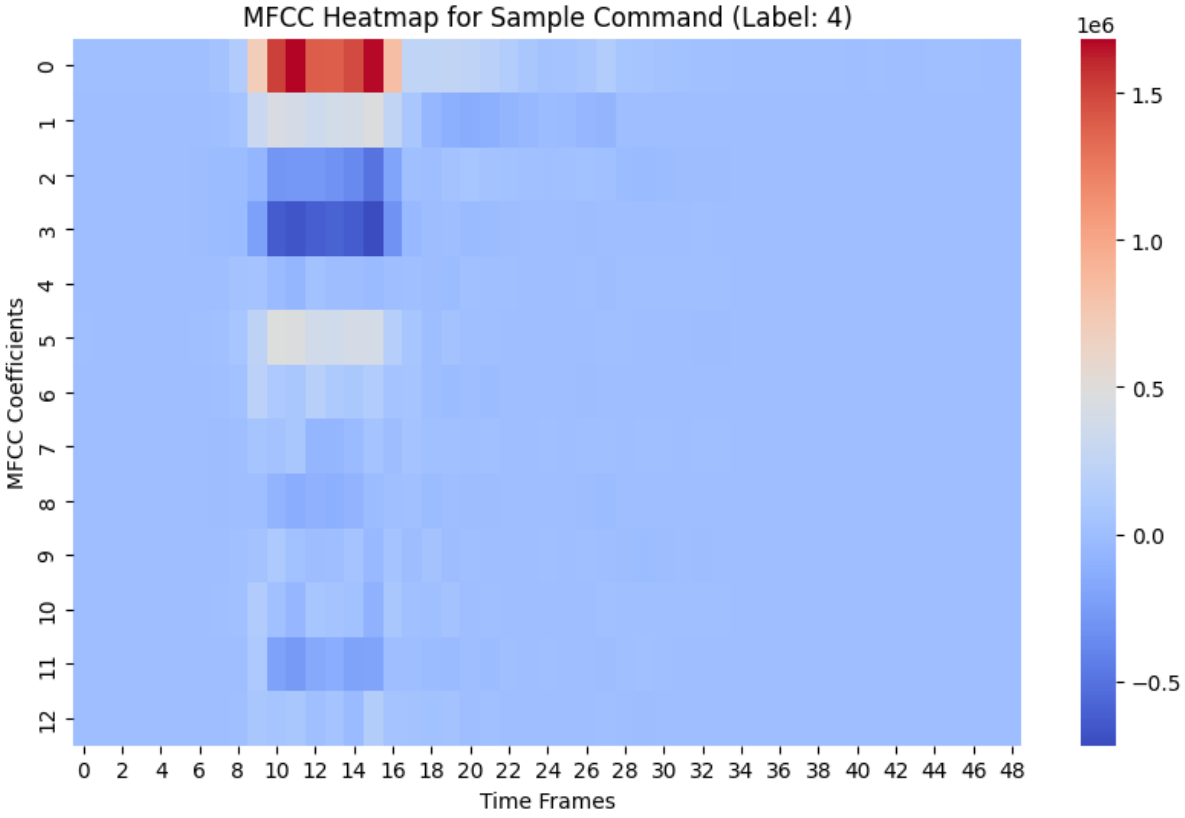
```python
waveform_first, *_ = train_set[0]
ipd.Audio(waveform_first.numpy(), rate=sample_rate)

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

0:00 / 0:01

Some snippets of analysis :




Audio Duration Distribution
Frequency
Duration (seconds)
160000
140000
120000
100000
80000
60000
40000
20000
0
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0


MFCC Heatmap for Sample Command (Label: 4)
MFCC Coefficients
Time Frames
1e6
1.5
1.0
0.5
0.0
−0.5
0 1 2 3 4 5 6 7 8 9 10 11 12
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48

Steps considered while setting up data loading and batching for training and testing a model using the SPEECHCOMMANDS dataset with PyTorch.

1. Padding Sequences (**pad_sequence** function)

- **Purpose**: Ensures all tensors in a batch are the same length by padding with zeros.
- **Details**:
  - Transposes each tensor in the batch.
  - Uses torch.nn.utils.rnn.pad_sequence to pad tensors to the same length.
  - Permutes the dimensions of the padded tensor to match the expected input shape for your model (usually (batch_size, channels, sequence_length)).

2. Collate Function (**collate_fn** function)

- **Purpose**: Defines how to collate (combine) individual data samples into a batch.
- **Details**:
  - Extracts waveforms and labels from each sample in the batch.
  - Applies pad_sequence to ensure waveforms in the batch have the same length.
  - Converts labels to indices using label_to_index and stacks them into a tensor.

3. DataLoader Configuration

- **Purpose**: Creates DataLoader instances for the training and test datasets.
- **Details**:
  - **train_loader**: Loads training data with batching, shuffling, and custom collate function.
    - batch_size=256
    - shuffle=True: Shuffles data at each epoch.
    - num_workers and pin_memory are set based on whether a GPU (cuda) or CPU is used.
  - **test_loader**: Loads test data with batching and custom collate function.
    - batch_size=256
    - shuffle=False: No shuffling for test data.
    - drop_last=False: Keeps the last batch even if it's smaller than the batch size.

Model Used (M5 Class)

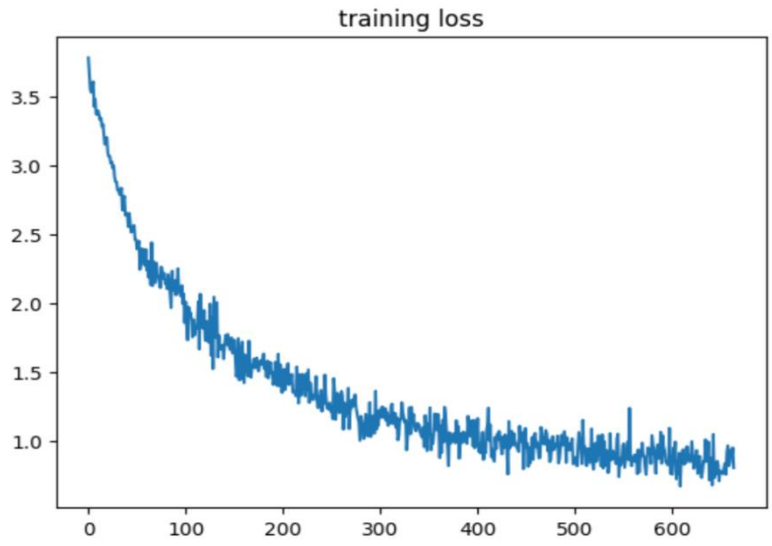
Purpose: Defines a CNN model for classification.
Components:

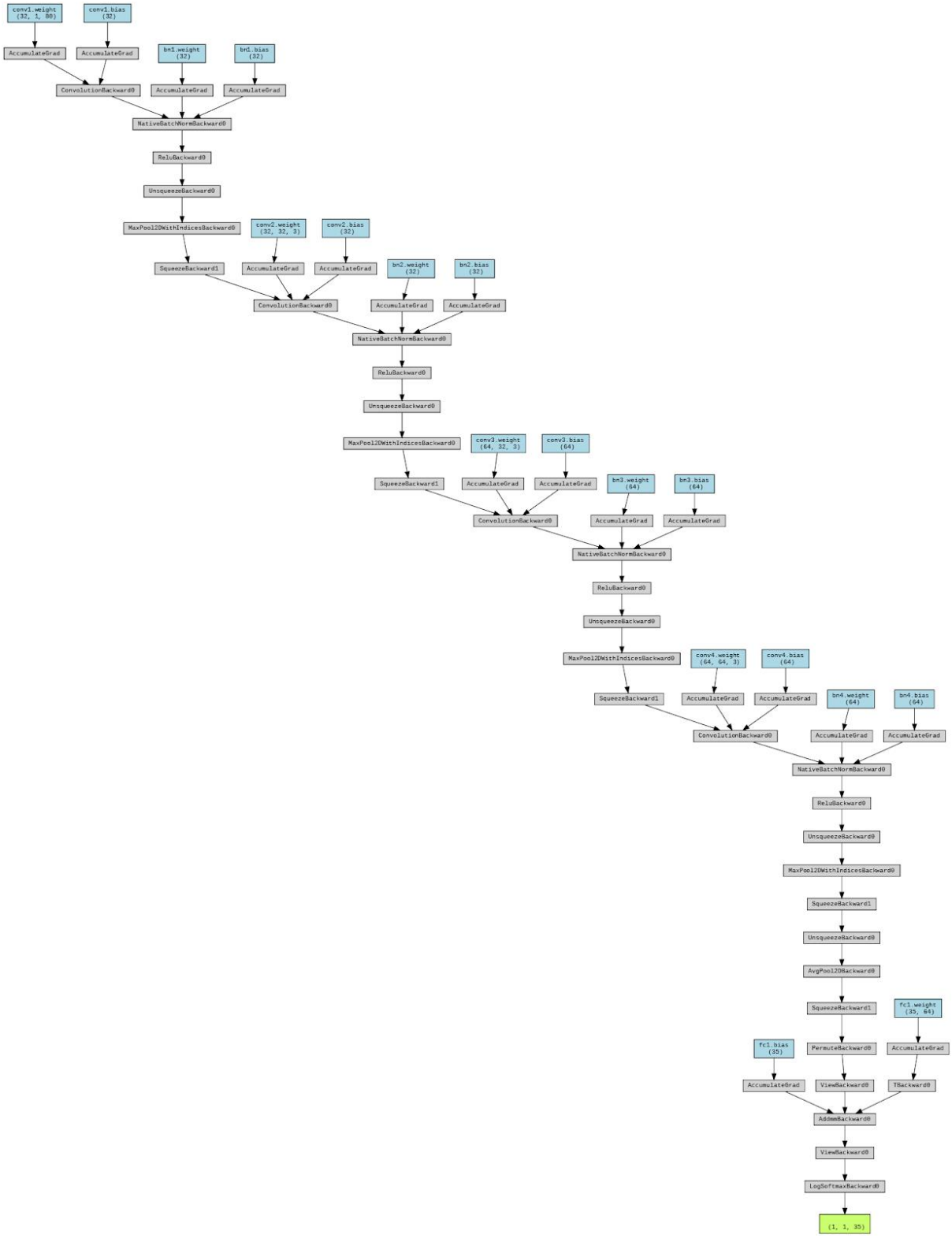
- **Convolutional Layers**:
  - conv1: 1D convolution with kernel size 80, stride 16.
  - conv2: 1D convolution with kernel size 3.
  - conv3: 1D convolution with kernel size 3, output channels doubled.
  - conv4: 1D convolution with kernel size 3, output channels doubled.
- **Batch Normalization**:
  - bn1, bn2, bn3, bn4: Normalize outputs of respective convolutional layers.
- **Pooling Layers**:
  - pool1, pool2, pool3, pool4: Max pooling with kernel size 4.
- **Fully Connected Layer**:
  - fc1: Linear layer that maps from the output channels to the number of classes.

- **Forward Pass**:
  - Applies convolutions, batch normalization, ReLU activation, pooling, and average pooling.
  - Reshapes and passes through a fully connected layer followed by log_softmax for classification.

```
 90%|█████████ | 1.8026666666666513/2 [17:05<01:44, 528.12s/it]Trai
 93%|█████████ | 1.8559999999999832/2 [17:35<01:13, 510.27s/it]Trai
100%|██████████| 1.9999999999999793/2 [18:54<00:00, 567.26s/it]
Test Epoch: 2   Accuracy: 5745/11005 (52%)
```

```
1 plt.plot(losses);
2 plt.title("training loss");
```




conv1.weight
(32, 1, 80)
conv1.bias
(32)
AccumulateGrad
AccumulateGrad
bn1.weight
(32)
bn1.bias
(32)
ConvolutionBackward0
AccumulateGrad
AccumulateGrad
NativeBatchNormBackward0
ReluBackward0
UnsqueezeBackward0
MaxPool2DWithIndicesBackward0
conv2.weight
(32, 32, 3)
conv2.bias
(32)
SqueezeBackward1
AccumulateGrad
AccumulateGrad
bn2.weight
(32)
bn2.bias
(32)
ConvolutionBackward0
AccumulateGrad
AccumulateGrad
NativeBatchNormBackward0
ReluBackward0
UnsqueezeBackward0
MaxPool2DWithIndicesBackward0
conv3.weight
(64, 32, 3)
conv3.bias
(64)
SqueezeBackward1
AccumulateGrad
AccumulateGrad
bn3.weight
(64)
bn3.bias
(64)
ConvolutionBackward0
AccumulateGrad
AccumulateGrad
NativeBatchNormBackward0
ReluBackward0
UnsqueezeBackward0
MaxPool2DWithIndicesBackward0
conv4.weight
(64, 64, 3)
conv4.bias
(64)
SqueezeBackward1
AccumulateGrad
AccumulateGrad
bn4.weight
(64)
bn4.bias
(64)
ConvolutionBackward0
AccumulateGrad
AccumulateGrad
NativeBatchNormBackward0
ReluBackward0
UnsqueezeBackward0
MaxPool2DWithIndicesBackward0
SqueezeBackward1
UnsqueezeBackward0
AvgPool2DBackward0
SqueezeBackward1
fc1.weight
(35, 64)
fc1.bias
(35)
PermuteBackward0
AccumulateGrad
AccumulateGrad
ViewBackward0
TBackward0
AddmmBackward0
ViewBackward0
LogSoftmaxBackward0
(1, 1, 35)