# Frame-Level Speech Recognition

## EXPERIMENT

- Build LA-DNN
  - A powerful model for speech recognition and good candidate to replace FF-DNN
  - Helps to remove the vanshing gradient problems in deep network
  - https://ieeexplore.ieee.org/document/7472646
- No Augmentation
- No hyper-parameter tuning however used our past knowledge to set hyper-paramters
- Advanced weight initialization, Label smoothing in loss function, gradient clipping and scheduler are used.

```
!pip install torch-summary --quiet
```

```
import torch
import numpy as np
import sklearn
import gc
import zipfile
import pandas as pd
from tqdm.auto import tqdm
import os
import datetime
import torchsummary
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print("Device: ", device)
```

```
Device:  cuda
```

```
# ### If you are using colab, you can import google drive t
from google.colab import drive
drive.mount('/content/drive')
```

```
!pip install --upgrade --force-reinstall --no-deps kaggle==
!mkdir -p /root/.kaggle
```

```
Collecting kaggle==1.5.8
  Downloading kaggle-1.5.8.tar.gz (59 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 59.2/59.2 kB 1.8 MB/s eta 0:00:
```

```
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: kaggle
  Building wheel for kaggle (setup.py) ... done
  Created wheel for kaggle: filename=kaggle-1.5.8-py3-none-any.whl size=73249
  Stored in directory: /root/.cache/pip/wheels/b5/23/bd/d33cbf399584fa44fa049
Successfully built kaggle
Installing collected packages: kaggle
  Attempting uninstall: kaggle
    Found existing installation: kaggle 1.7.4.2
    Uninstalling kaggle-1.7.4.2:
      Successfully uninstalled kaggle-1.7.4.2
Successfully installed kaggle-1.5.8
```

```
with open("/root/.kaggle/kaggle.json", "w+") as f:
    f.write('{"username":"dineshbuswala","key":"a67fefaecac

!chmod 600 /root/.kaggle/kaggle.json
```

```
# commands to download data from kaggle

!kaggle competitions download -c 11785-hw1p2-f23 --force
!mkdir -p '/content/data'

# !unzip -qo /content/11785-hw1p2-f23.zip -d '/content/data
```

```
Downloading 11785-hw1p2-f23.zip to /kaggle/working
100%|██████████| 3.99G/3.99G [00:24<00:00, 219MB
100%|██████████| 3.99G/3.99G [00:24<00:00, 174MB
```

```
!unzip -qo /kaggle/working/11785-hw1p2-f23.zip -d '/content
```

```
# ! rm -r /content/data/11-785-f23-hw1p2/test-clean
# ! rm -r /content/11785-hw1p2-f23.zip
```

```
 ! ls /content/data/11-785-f23-hw1p2/train-clean-100/mfcc/
```

```
28539
```

```
### configuration variables
EPOCHS                          = 4
BATCH_SIZE                      = 2048 * 2
LEFT_CONTEXT                    = 7
RIGHT_CONTEXT                   = 7
INITIAL_LEARNING_RATE           = 1e-3
L2_PENALTY                      = 1e-5
STEP_SIZE                       = 2
```

```
GAMMA                           = 0.1
BASE_DIRECTORY                  = '/content/data/11-785-f23
TRAINING_DATA                   = BASE_DIRECTORY + 'train-c
EVALUATION_DATA                 = BASE_DIRECTORY + 'dev-cle
PHONEMES                        = ['[SIL]',   'AA',    'AE'
                                     'B',     'CH',    'D',
                                     'F',     'G',     'HH',
                                     'L',     'M',     'N',
                                     'R',     'S',     'SH',
                                     'V',     'W',     'Y',
PHONEMES_TO_INDEX                = {phoneme: idx for idx, p
NUMBER_OF_NEURONS                = [2048, 2048, 1024, 1024,
MODEL_DIR                        = "/content"
CLIP_VALUE                       = 1.0
LABEL_SMOOTHING                  = 0.01
```

```
class AudioDataset(torch.utils.data.Dataset):
    def __init__(self, root, phonemes = PHONEMES_TO_INDEX,
                 left_context = LEFT_CONTEXT,
                 right_context = RIGHT_CONTEXT):

        self.left_context = left_context
        self.right_context = right_context
        self.phonemes_mapping = phonemes

        self.mfcc_dir = os.path.join(root, 'mfcc')
        self.transcript_dir = os.path.join(root, 'transcri

        # List and sort mfcc and transcript files
        mfcc_names = sorted(os.listdir(self.mfcc_dir))
        transcript_names = sorted(os.listdir(self.transcri

        # Sanity check
        assert len(mfcc_names) == len(transcript_names), "

        total_frames = 0

        for i in range(len(mfcc_names)):
            mfcc_path = os.path.join(self.mfcc_dir, mfcc_n

            mfcc = np.load(mfcc_path,
                           allow_pickle = False,
                           mmap_mode='r')
            total_frames += mfcc.shape[0]
            del mfcc
```

```
30
31          sample_mfcc = np.load(os.path.join(self.mfcc_dir,
32          self.mfcc_dim = sample_mfcc.shape[1]
33          ### Right Padding is added here automatically
34          self.mfccs = np.zeros((total_frames + right_contex
35          self.transcripts = [None] * (total_frames + right_
36          ### Release memory
37          del sample_mfcc, total_frames
38          gc.collect()
39
40          index = 0
41          for i in range(len(mfcc_names)):
42              mfcc = np.load(os.path.join(self.mfcc_dir, mfc
43                             allow_pickle = False,
44                             mmap_mode = 'r')
45
46              mfcc = (mfcc - mfcc.mean(axis = 1, keepdims =
47
48              transcript = np.load(os.path.join(self.transcr
49                                   allow_pickle = False,
50                                   mmap_mode='r')[1:-1]
51
52              self.mfccs[index: index + mfcc.shape[0]] = mfc
53              self.transcripts[index: index + len(transcript
54              index += mfcc.shape[0]
55              del mfcc, transcript
56
57              if i % 1000 == 0:
58                  gc.collect() ### Release memory
59
60          # Save original dataset length (before adding righ
61          self.length = len(self.mfccs) - right_context
62
63          # Map transcript phonemes to indices
64          self.transcripts = [self.phonemes_mapping.get(p, -
65
66          ### Release memory
67          del mfcc_names, transcript_names
68          gc.collect()
69
70      def __len__(self):
71          return self.length
72
73      def __getitem__(self, ind):
74          if ind < self.left_context:  # index is less than
```

```
            # zeros need to prepend with it
            padding = []
            for _ in range(self.left_context - ind):
                padding.append(np.zeros((1, self.mfcc_dim)

            for i in range(ind):
                padding.append(self.mfccs[i][None, :])

            # Include current and right context frames
            current = self.mfccs[ind][None, :]  # Ensure s
            right_context = [self.mfccs[i][None, :] for i
            frames = np.concatenate(padding + [current] +

        else:  # when index is greater than or equal to le
            left_context_frames = self.mfccs[ind - self.le
            current = self.mfccs[ind][None, :]  # Ensure s
            right_context_frames = self.mfccs[ind + 1: ind

            frames = np.concatenate([left_context_frames,

        frames = frames.flatten()  # Flatten to get 1D dat
        frames = torch.FloatTensor(frames)  # Convert to t
        phonemes = torch.tensor(self.transcripts[ind], dty

        return frames, phonemes

```

```
# Create a dataset object using the AudioDataset class for
train_data = AudioDataset(TRAINING_DATA)

# Create a dataset object using the AudioDataset class for
val_data = AudioDataset(EVALUATION_DATA)

```

```
train_loader = torch.utils.data.DataLoader(
    dataset     = train_data,
    num_workers = 2,
    batch_size  = BATCH_SIZE,
    pin_memory  = True,
    shuffle     = True
)

val_loader = torch.utils.data.DataLoader(
    dataset     = val_data,
    num_workers = 1,
```

```
    batch_size  = BATCH_SIZE,
    pin_memory  = True,
    shuffle     = False
)

print("Train dataset samples = {}, batches = {}".format(tra
print("Validation dataset samples = {}, batches = {}".forma
```

```
Train dataset samples = 36091157, batches = 8812
Validation dataset samples = 1928204, batches = 471
```

```
# # Testing code to check if data loaders are working as ex
# total_batches = len(train_loader)

# for i, (frames, phoneme) in enumerate(train_loader):
#     if i < 10 or i >= total_batches - 10:
#         print(f"Batch {i + 1}/{total_batches}")
#         print("Frames:", frames)
#         print("Phoneme:", phoneme)
#         print("-" * 50)

```

```
class LADNNLayer(torch.nn.Module):
    def __init__(self, in_features, out_features, rank = -1
        super(LADNNLayer, self).__init__()

        self.in_features = in_features
        self.out_features = out_features
        self.activation = torch.nn.ReLU()
        if rank == -1:
            self.intermediate = in_features // 2
        else:
            self.intermediate = rank
        self.U = torch.nn.Linear(self.in_features, self.int
        self.V = torch.nn.Linear(self.intermediate, self.ou
        self.T = torch.nn.Linear(self.in_features, self.out

        # Apply weight initialization
        self._initialize_weights()

    def _initialize_weights(self):
        print('Initialization of weights using Kaiming')
        for m in [self.U] + [self.V] + [self.T]:
            if isinstance(m, torch.nn.Linear):
                # Kaiming initialization for weights
```

```
                torch.nn.init.kaiming_uniform_(m.weight, no

    def forward(self, x):
        t_x = self.T(x)
        u_x = self.U(x)
        v_x = self.V(self.activation(u_x))
        return v_x + t_x



class Network(torch.nn.Module):
    def __init__(self, input_size):
        super(Network, self).__init__()

        # Neurons in each layer: input -> hidden(s) -> outp
        self.neurons = [input_size] + NUMBER_OF_NEURONS + [

        layers = []
        for in_features, out_features in zip(self.neurons[:
            layers.append(LADNNLayer(in_features, out_featu

        # Final layer
        layers.append(LADNNLayer(self.neurons[-2], self.neu

        # Combine all into a sequential model
        self.model = torch.nn.Sequential(*layers)

        # Apply weight initialization
        self._initialize_weights()

    def _initialize_weights(self):
        print('Initialization of weights using Kaiming')
        for m in self.model:
            if isinstance(m, torch.nn.Linear):
                # Kaiming initialization for weights
                torch.nn.init.kaiming_uniform_(m.weight, no


    def forward(self, x):
        out = self.model(x)
        return out

INPUT_SIZE = (LEFT_CONTEXT + RIGHT_CONTEXT + 1) * 28
model = Network(INPUT_SIZE).to(device)
```

```
# Pass the input size as a tuple, without the batch dimensi
torchsummary.summary(model, (INPUT_SIZE,))
```

```
Initialization of weights using Kaiming
Initialization of weights using Kaiming
Initialization of weights using Kaiming
Initialization of weights using Kaiming
Initialization of weights using Kaiming
Initialization of weights using Kaiming
Initialization of weights using Kaiming
Initialization of weights using Kaiming
Initialization of weights using Kaiming
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 2048]         862,208
            Linear-2                 [-1, 210]          88,410
              ReLU-3                 [-1, 210]               0
            Linear-4                [-1, 2048]         432,128
         LADNNLayer-5                [-1, 2048]               0
            Linear-6                [-1, 2048]       4,196,352
            Linear-7                [-1, 1024]       2,098,176
              ReLU-8                [-1, 1024]               0
            Linear-9                [-1, 2048]       2,099,200
        LADNNLayer-10                [-1, 2048]               0
           Linear-11                [-1, 1024]       2,098,176
           Linear-12                [-1, 1024]       2,098,176
             ReLU-13                [-1, 1024]               0
           Linear-14                [-1, 1024]       1,049,600
        LADNNLayer-15                [-1, 1024]               0
           Linear-16                [-1, 1024]       1,049,600
           Linear-17                 [-1, 512]         524,800
             ReLU-18                 [-1, 512]               0
           Linear-19                [-1, 1024]         525,312
        LADNNLayer-20                [-1, 1024]               0
           Linear-21                 [-1, 512]         524,800
           Linear-22                 [-1, 512]         524,800
             ReLU-23                 [-1, 512]               0
           Linear-24                 [-1, 512]         262,656
        LADNNLayer-25                 [-1, 512]               0
           Linear-26                 [-1, 256]         131,328
           Linear-27                 [-1, 256]         131,328
             ReLU-28                 [-1, 256]               0
           Linear-29                 [-1, 256]          65,792
        LADNNLayer-30                 [-1, 256]               0
           Linear-31                 [-1, 256]          65,792
           Linear-32                 [-1, 128]          32,896
             ReLU-33                 [-1, 128]               0
           Linear-34                 [-1, 256]          33,024
        LADNNLayer-35                 [-1, 256]               0
           Linear-36                  [-1, 40]          10,280
           Linear-37                 [-1, 128]          32,896
             ReLU-38                 [-1, 128]               0
           Linear-39                  [-1, 40]           5,160
        LADNNLayer-40                  [-1, 40]               0
================================================================
Total params: 18,942,890
Trainable params: 18,942,890
Non-trainable params: 0
----------------------------------------------------------------
```

Input size (MB): 0.00

```
criterion = torch.nn.CrossEntropyLoss(label_smoothing = LAB
# We use CE because the task is multi-class classification

optimizer = torch.optim.Adam(model.parameters(),
                             lr = INITIAL_LEARNING_RATE,
                             weight_decay = L2_PENALTY) #De
scheduler = torch.optim.lr_scheduler.StepLR(
    optimizer, step_size = STEP_SIZE, gamma = GAMMA
)
# Refer - https://pytorch.org/docs/stable/notes/amp_example
```

```
torch.cuda.empty_cache()
gc.collect()
```

```
101
```

```
def train(model, dataloader, optimizer, criterion):

    model.train()
    tloss, tacc = 0, 0 # Monitoring loss and accuracy
    batch_bar   = tqdm(total=len(train_loader), dynamic_nc

    for i, (frames, phonemes) in enumerate(dataloader):

        ### Initialize Gradients
        optimizer.zero_grad()

        ### Move Data to Device (Ideally GPU)
        frames      = frames.to(device)
        phonemes    = phonemes.to(device)

        ### Forward Propagation
        logits  = model(frames)

        ### Loss Calculation
        loss    = criterion(logits, phonemes)

        ### Backward Propagation
        loss.backward()

        ### Clip gradients
        torch.nn.utils.clip_grad_norm_(model.parameters(),

        ### Gradient Descent
```

```
        optimizer.step()

        tloss   += loss.item()
        tacc    += torch.sum(torch.argmax(logits, dim= 1) =

        batch_bar.set_postfix(loss="{:.04f}".format(float(t
                              acc="{:.04f}%".format(float(t
        batch_bar.update()

        ### Release memory
        del frames, phonemes, logits
        torch.cuda.empty_cache()

    batch_bar.close()
    tloss   /= len(train_loader)
    tacc    /= len(train_loader)

    return tloss, tacc


def eval(model, dataloader):

    model.eval() # set model in evaluation mode
    vloss, vacc = 0, 0 # Monitoring loss and accuracy
    batch_bar   = tqdm(total=len(val_loader), dynamic_ncols

    for i, (frames, phonemes) in enumerate(dataloader):

        ### Move data to device (ideally GPU)
        frames      = frames.to(device)
        phonemes    = phonemes.to(device)

        # makes sure that there are no gradients computed a
        with torch.inference_mode():
            ### Forward Propagation
            logits  = model(frames)
            ### Loss Calculation
            loss    = criterion(logits, phonemes)

        vloss   += loss.item()
        vacc    += torch.sum(torch.argmax(logits, dim= 1) =


        batch_bar.set_postfix(loss="{:.04f}".format(float(v
                              acc="{:.04f}%".format(float(v
        batch_bar.update()
```

```
        ### Release memory
        del frames, phonemes, logits
        torch.cuda.empty_cache()

    batch_bar.close()
    vloss   /= len(val_loader)
    vacc    /= len(val_loader)

    return vloss, vacc
```

```
best_model_path = os.path.join(MODEL_DIR, "best_model.pt")
best_acc = -np.inf
for epoch in range(EPOCHS):
    ### clean up memory before computation
    torch.cuda.empty_cache()
    gc.collect()


    print(f"\nEpoch {epoch + 1}/{EPOCHS}")

    curr_lr = float(optimizer.param_groups[0]['lr'])
    train_loss, train_acc = train(model, train_loader, opti
    val_loss, val_acc = eval(model, val_loader)

    print(f"\tTrain Acc: {train_acc*100:.2f}%\tTrain Loss:
    print(f"\tVal   Acc: {val_acc*100:.2f}%\tVal   Loss: {v

    # Save model at every epoch
    epoch_model_path = os.path.join(MODEL_DIR, f"model_at_e
    torch.save(model.state_dict(), epoch_model_path)

    # Save best model
    if val_acc > best_acc:
        best_acc = val_acc
        torch.save(model.state_dict(), best_model_path)
        print(f"Updated Best Model at: {best_model_path}")

    ### take step in adjusting the learning rate
    scheduler.step()


```

```
Epoch 1/4
Train:   0%|          | 0/8812 [00:00<?, ?it/s]
Val:   0%|          | 0/471 [00:00<?, ?it/s]
        Train Acc: 67.04%       Train Loss: 13.0031     LR: 0.0010000
        Val   Acc: 69.21%       Val   Loss: 1.0357
Updated Best Model at: /content/best_model.pt

Epoch 2/4
Train:   0%|          | 0/8812 [00:00<?, ?it/s]
Val:   0%|          | 0/471 [00:00<?, ?it/s]
        Train Acc: 72.84%       Train Loss: 0.9069      LR: 0.0010000
        Val   Acc: 70.96%       Val   Loss: 0.9690
Updated Best Model at: /content/best_model.pt

Epoch 3/4
Train:   0%|          | 0/8812 [00:00<?, ?it/s]
Val:   0%|          | 0/471 [00:00<?, ?it/s]
        Train Acc: 77.41%       Train Loss: 0.7553      LR: 0.0001000
        Val   Acc: 74.43%       Val   Loss: 0.8521
Updated Best Model at: /content/best_model.pt

Epoch 4/4
Train:   0%|          | 0/8812 [00:00<?, ?it/s]
Val:   0%|          | 0/471 [00:00<?, ?it/s]
        Train Acc: 78.37%       Train Loss: 0.7237      LR: 0.0001000
        Val   Acc: 74.58%       Val   Loss: 0.8479
Updated Best Model at: /content/best_model.pt
```

```
# Load best model after training
model.load_state_dict(torch.load('/content/best_model.pt'))

```

```
<All keys matched successfully>
```

```
from sklearn.metrics import classification_report, confusio
model.eval()  # Set model in evaluation mode
predicted = []
groundtruth = []

for frames, phonemes in val_loader:

    # Move data to device
    frames = frames.to(device)
    phonemes = phonemes.to(device)

    # Disable gradient calculation
    with torch.inference_mode():
        logits = model(frames)

    predict = torch.argmax(logits, dim = 1)

```

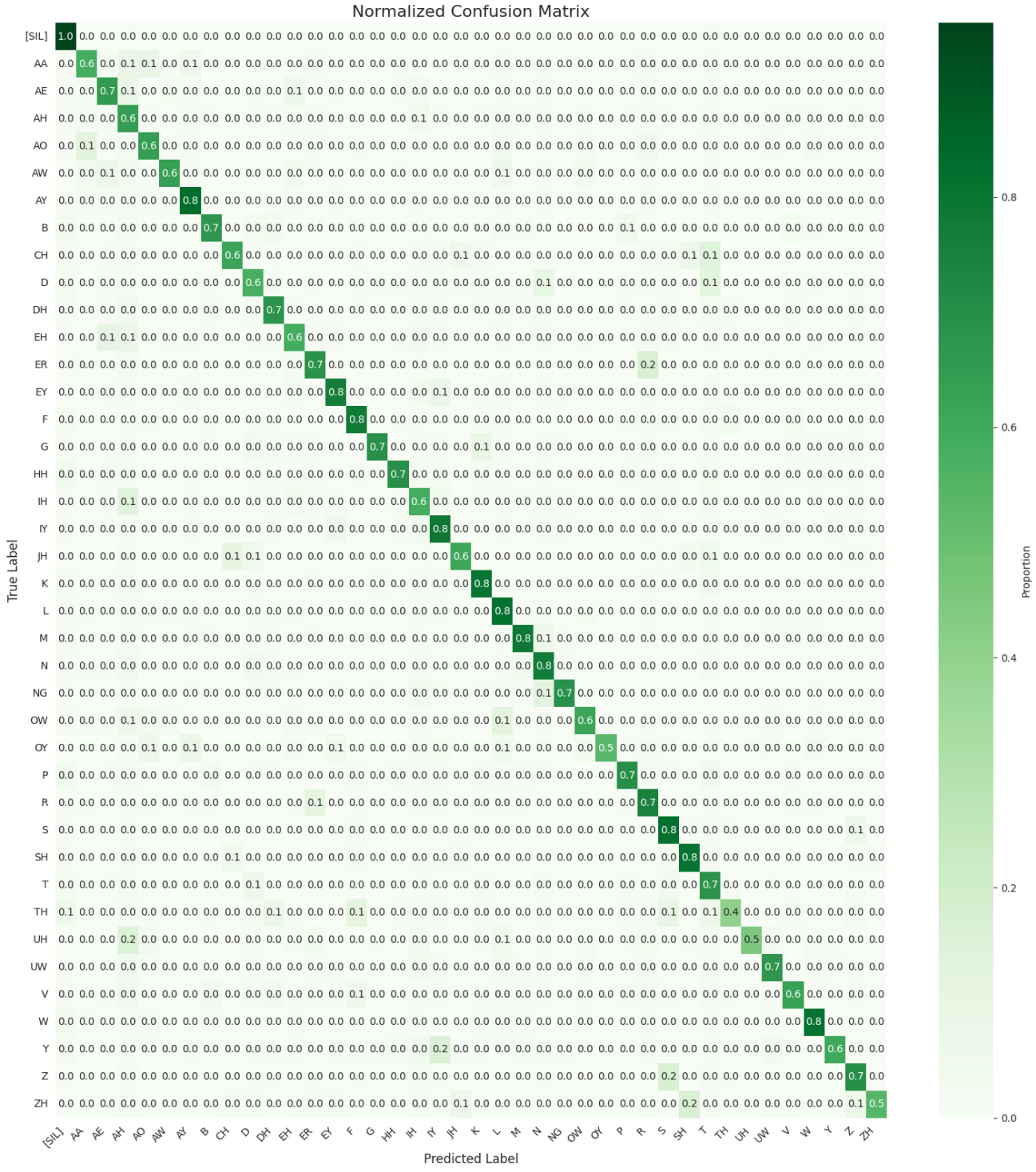
```
    # Detach and move to CPU for evaluation
    predicted.extend(predict.detach().cpu().tolist())
    groundtruth.extend(phonemes.detach().cpu().tolist())

    # Release memory
    del frames, phonemes, logits, predict
    torch.cuda.empty_cache()

# Print classification report
print(classification_report(
    groundtruth,
    predicted,
    target_names = PHONEMES  # Skipping SOS and EOS tokens
))

```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| [SIL] | 0.94 | 0.95 | 0.94 | 319908 |
| AA | 0.59 | 0.59 | 0.59 | 29688 |
| AE | 0.65 | 0.66 | 0.66 | 49298 |
| AH | 0.61 | 0.64 | 0.63 | 123734 |
| AO | 0.66 | 0.64 | 0.65 | 29340 |
| AW | 0.72 | 0.63 | 0.67 | 20274 |
| AY | 0.79 | 0.83 | 0.81 | 49332 |
| B | 0.69 | 0.68 | 0.68 | 23607 |
| CH | 0.65 | 0.62 | 0.63 | 12644 |
| D | 0.65 | 0.58 | 0.61 | 62763 |
| DH | 0.70 | 0.68 | 0.69 | 37100 |
| EH | 0.62 | 0.59 | 0.60 | 47112 |
| ER | 0.69 | 0.68 | 0.69 | 54928 |
| EY | 0.75 | 0.76 | 0.76 | 36184 |
| F | 0.72 | 0.78 | 0.75 | 37562 |
| G | 0.74 | 0.69 | 0.71 | 13541 |
| HH | 0.75 | 0.70 | 0.72 | 34813 |
| IH | 0.63 | 0.59 | 0.61 | 74887 |
| IY | 0.77 | 0.79 | 0.78 | 70861 |
| JH | 0.69 | 0.61 | 0.65 | 8730 |
| K | 0.77 | 0.80 | 0.79 | 47016 |
| L | 0.73 | 0.82 | 0.77 | 65902 |
| M | 0.77 | 0.78 | 0.78 | 44728 |
| N | 0.74 | 0.77 | 0.75 | 94541 |
| NG | 0.73 | 0.69 | 0.71 | 19327 |
| OW | 0.71 | 0.62 | 0.66 | 30755 |
| OY | 0.77 | 0.54 | 0.63 | 3861 |
| P | 0.75 | 0.70 | 0.72 | 34131 |
| R | 0.69 | 0.75 | 0.72 | 62686 |
| S | 0.80 | 0.82 | 0.81 | 101184 |
| SH | 0.81 | 0.81 | 0.81 | 17628 |
| T | 0.69 | 0.68 | 0.68 | 97390 |
| TH | 0.44 | 0.40 | 0.42 | 9247 |
| UH | 0.64 | 0.45 | 0.53 | 6286 |
| UW | 0.73 | 0.67 | 0.70 | 26691 |
| V | 0.71 | 0.63 | 0.67 | 27440 |
| W | 0.78 | 0.81 | 0.80 | 37697 |

```
              Y       0.70      0.60      0.65      9669
              Z       0.71      0.70      0.70     54850
             ZH       0.71      0.55      0.62       869

       accuracy                           0.75   1928204
      macro avg       0.71      0.68      0.69   1928204
   weighted avg       0.74      0.75      0.74   1928204
```

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_style("darkgrid")
```

```
# Compute confusion matrix
cm = confusion_matrix(groundtruth, predicted)

# Normalize confusion matrix by row (i.e., by true labels)
cm_normalized = cm.astype('float') / cm.sum(axis = 1, keepd

# Replace NaNs (from division by zero, if any row sum is 0)
cm_normalized = np.nan_to_num(cm_normalized)

# Plot
plt.figure(figsize=(15, 16))
sns.heatmap(cm_normalized,
            annot=True,
            fmt=".1f",
            cmap="Greens",
            xticklabels=PHONEMES,
            yticklabels=PHONEMES,
            cbar_kws={'label': 'Proportion'})

plt.title('Normalized Confusion Matrix', fontsize=16)
plt.xlabel('Predicted Label', fontsize=12)
plt.ylabel('True Label', fontsize=12)
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.yticks(rotation=0, fontsize=10)
plt.tight_layout()
plt.show()

```


Normalized Confusion Matrix
True Label
Predicted Label
Proportion
[SIL] AA AE AH AO AW AY B CH D DH EH ER EY F G HH IH IY JH K L M N NG OW OY P R S SH T TH UH UW V W Y Z ZH
0.0
0.2
0.4
0.6
0.8