Task description

- Classify the speakers of given features.
- Main goal: Learn how to use transformer.
- Baselines:
  - Easy: Run sample code and know how to use transformer.
  - Medium: Know how to adjust parameters of transformer.
  - Hard: Construct conformer which is a variety of transformer.
- Other links
  - Kaggle: link
  - Slide: link
  - Data: link
  - Video (Chinese): link
  - Video (English): link
  - Solution for downloading dataset fail.: link

Download dataset

- Please follow here to download data
- Data is here

```
from google.colab import drive
drive.mount("/content/drive")
```

```
Mounted at /content/drive
```

```
!gdown --id '1X5jTD1ZQ9rWWyjMc4qHZeDc1XNuXUt1C' --output Dataset.zip
!unzip Dataset.zip
```

```
Streaming output truncated to the last 5000 lines.
  inflating: Dataset/uttr-302451c82acc472ea4dbfe7e09104752.pt
  inflating: Dataset/uttr-13307b1f79fa4c34ae0ac7dd01e550d0.pt
  inflating: Dataset/uttr-addf4e847665455da803a433a0a872f4.pt
  inflating: Dataset/uttr-2f0b654391a44cb3b4335bef62ad7a5a.pt
  inflating: Dataset/uttr-1f05e4d6775144378000b9ffd7ff907f.pt
  inflating: Dataset/uttr-fbae633b89044403a6c056987df8c720.pt
  inflating: Dataset/uttr-6d67740c09c44ccb95541e1fd389d205.pt
  inflating: Dataset/uttr-c9bae6645d62413ea745b8e0580af07c.pt
  inflating: Dataset/uttr-f636fd38b3c2409ab9e760773e9052f4.pt
  inflating: Dataset/uttr-1ef19a9560d44a86a2edbf5c7f21b986.pt
  inflating: Dataset/uttr-92ca62f99d894c789d1c5cda1e198b27.pt
  inflating: Dataset/uttr-dc1fc195c84d49bfba41c82522423498.pt
  inflating: Dataset/uttr-902f895bc2a9476084b4af902d4ce297.pt
  inflating: Dataset/uttr-e4f7b5f8565e4463adb962a24536509a.pt
  inflating: Dataset/uttr-3a6a4a50ff56419fb4a5c1688d117203.pt
  inflating: Dataset/uttr-6a0f2bf7a8314cc3b03bb76c1f108714.pt
  inflating: Dataset/uttr-1952aba475b549ce90babd2789b01c87.pt
  inflating: Dataset/uttr-3320c35117fb4065b760f6e8ef4e196e.pt
  inflating: Dataset/uttr-9dc19bc443cc4445ab792f07992985b3.pt
  inflating: Dataset/uttr-aa7761f8234644b1a67a867727aea3c1.pt
  inflating: Dataset/uttr-2d0b869c6c16466ca00e0d40079fc5a2.pt
  inflating: Dataset/uttr-5ad9969e3a1d49439ac5da7fb480b952.pt
  inflating: Dataset/uttr-7a0092616ba148dfb36ec325777cf372.pt
  inflating: Dataset/uttr-f251cde008c84eb98caa8854b51f6d29.pt
  inflating: Dataset/uttr-f6b21a7eee9149ba9bc6ee551835c9ff.pt
  inflating: Dataset/uttr-1ac3db9154694ec5a71183da9c6b4851.pt
  inflating: Dataset/uttr-348c7bb1730344f18f6c70af5b0fccda.pt
  inflating: Dataset/uttr-e6c47a0fae064f14822600816bc3baf2.pt
  inflating: Dataset/uttr-72192bf8e0c04a8f8d3330aa7f5729d6.pt
  inflating: Dataset/uttr-c366e9e0a9fc477e88607448b87f5feb.pt
  inflating: Dataset/uttr-76db32f54635458e8e801ccda15a7336.pt
  inflating: Dataset/uttr-03a6ec6a2e5c4dc48c3c11d4aef21984.pt
  inflating: Dataset/uttr-954195089da24df9a8165ecfaafe53cd.pt
  inflating: Dataset/uttr-00232cb6f69547bfbbb75dbcf7ccd881.pt
  inflating: Dataset/uttr-ae90911d7cea413e8b85c4a82d8a2c85.pt
```

```
inflating: Dataset/uttr-9f9908469ccd436a8314551fee63c0c9.pt
inflating: Dataset/uttr-74ceac84c0df483f87a4e00c348a7244.pt
inflating: Dataset/uttr-96bbcc39c2f24b7f8b746ad9e5d786e3.pt
inflating: Dataset/uttr-f48bbb8d440a48b8aee7aa49474df300.pt
inflating: Dataset/uttr-ff13d81b70a74e44a1799cb19c8389ab.pt
inflating: Dataset/uttr-21f4a91f45da49aaa630c45404ffb708.pt
inflating: Dataset/uttr-2a657fb7c93346c49f5f96aafd76cd91.pt
inflating: Dataset/uttr-c2baead04e41445298376616b7e3a714.pt
inflating: Dataset/uttr-e174a1e1b55a4d4daa65f3a03bd7e8cd.pt
inflating: Dataset/uttr-7a7e6523d8d84afd9ef0488d21d66b62.pt
inflating: Dataset/uttr-9e0cea0638354c2ca8efe876a654f615.pt
inflating: Dataset/uttr-0b6e721563e6416c99bf3d05f2d695bd.pt
inflating: Dataset/uttr-1eca4be036504ec0a43a4fcfc3402722.pt
inflating: Dataset/uttr-414bb752d9034cf3ad6e46c2e556174c.pt
inflating: Dataset/uttr-0c99f7d3c6a945aaa60dc14ca41cc8b1.pt
inflating: Dataset/uttr-b0d56ce3602246cc98e9bbbabded87c8.pt
inflating: Dataset/uttr-78313aca52ad41d4a827a5dfcbff63a3.pt
inflating: Dataset/uttr-bce6d41ad78946f4a4a8b88846eef37e.pt
inflating: Dataset/uttr-d564f1bed8224612ab81b5690d2836fe.pt
inflating: Dataset/uttr-d3d739d50a36494faea91b983354119f.pt
inflating: Dataset/uttr-3e2e846cf3c34939ad0c7cd49640bb5e.pt
inflating: Dataset/uttr-29f5ca0117d8405093b793d886899457.pt
```

```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
    print('Not connected to a GPU')
else:
    print(gpu_info)
```

```
Mon Feb 19 07:45:46 2024
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  Tesla V100-SXM2-16GB           Off | 00000000:00:04.0 Off |                    0 |
| N/A   32C    P0              22W / 300W |      0MiB / 16384MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|  No running processes found                                                           |
+---------------------------------------------------------------------------------------+
```

Data

Dataset

- Original dataset is Voxceleb1.
- The license and complete version of Voxceleb1.
- We randomly select 600 speakers from Voxceleb1.
- Then preprocess the raw waveforms into mel-spectrograms.
- Args:
  - data_dir: The path to the data directory.
  - metadata_path: The path to the metadata.
  - segment_len: The length of audio segment for training.
- The architecture of data directory
  - data directory
    |---- metadata.json

|---- testdata.json
|---- mapping.json
|---- uttr-{random string}.pt

- The information in metadata
  - "n_mels": The dimention of mel-spectrogram.
  - "speakers": A dictionary.
    - Key: speaker ids.
    - value: "feature_path" and "mel_len"

For efficiency, we segment the mel-spectrograms into segments in the traing step.

```
import os
import json
import torch
import random
from pathlib import Path
from torch.utils.data import Dataset
from torch.nn.utils.rnn import pad_sequence

# default segment_len=128
class myDataset(Dataset):
    def __init__(self, data_dir, segment_len=128):
        self.data_dir = data_dir
        self.segment_len = segment_len

        # Load the mapping from speaker neme to their corresponding id.
        mapping_path = Path(data_dir) / "mapping.json"
        mapping = json.load(mapping_path.open())
        self.speaker2id = mapping["speaker2id"]

        # Load metadata of training data.
        metadata_path = Path(data_dir) / "metadata.json"
        metadata = json.load(open(metadata_path))["speakers"]

        # Get the total number of speaker.
        self.speaker_num = len(metadata.keys())
        self.data = []
        for speaker in metadata.keys():
            for utterances in metadata[speaker]:
                self.data.append([utterances["feature_path"], self.speaker2id[speaker]])

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        feat_path, speaker = self.data[index]
        # Load preprocessed mel-spectrogram.
        mel = torch.load(os.path.join(self.data_dir, feat_path))

        # Segmemt mel-spectrogram into "segment_len" frames.
        if len(mel) > self.segment_len:
            # Randomly get the starting point of the segment.
            start = random.randint(0, len(mel) - self.segment_len)
            # Get a segment with "segment_len" frames.
            mel = torch.FloatTensor(mel[start:start+self.segment_len])
        else:
            mel = torch.FloatTensor(mel)
        # Turn the speaker id into long for computing loss later.
        speaker = torch.FloatTensor([speaker]).long()
        return mel, speaker

    def get_speaker_number(self):
        return self.speaker_num
```

Dataloader

- Split dataset into training dataset(90%) and validation dataset(10%).
- Create dataloader to iterate the data.

```python
import torch
from torch.utils.data import DataLoader, random_split
from torch.nn.utils.rnn import pad_sequence


def collate_batch(batch):
    # Process features within a batch.
    #. zip(*) 可理解為解壓縮
    """Collate a batch of data."""
    mel, speaker = zip(*batch)
    # Because we train the model batch by batch, we need to pad the features in the same batch to make their lengths the same.
    # 將mel對齊成一樣長度
    mel = pad_sequence(mel, batch_first=True, padding_value=-20)    # pad log 10^(-20) which is very small value.
    # mel: (batch size, length, 40)
    return mel, torch.FloatTensor(speaker).long()


def get_dataloader(data_dir, batch_size, n_workers):
    """Generate dataloader"""
    dataset = myDataset(data_dir)
    speaker_num = dataset.get_speaker_number()
    # Split dataset into training dataset and validation dataset
    trainlen = int(0.9 * len(dataset))
    lengths = [trainlen, len(dataset) - trainlen]
    trainset, validset = random_split(dataset, lengths)

    train_loader = DataLoader(
        trainset,
        batch_size=batch_size,
        shuffle=True,
        drop_last=True,
        num_workers=n_workers,
        pin_memory=True,
        collate_fn=collate_batch,
    )
    valid_loader = DataLoader(
        validset,
        batch_size=batch_size,
        num_workers=n_workers,
        drop_last=True,
        pin_memory=True,
        collate_fn=collate_batch,
    )

    return train_loader, valid_loader, speaker_num


# pip install conformer
```

```
Collecting conformer
  Downloading conformer-0.3.2-py3-none-any.whl (4.3 kB)
Collecting einops>=0.6.1 (from conformer)
  Downloading einops-0.7.0-py3-none-any.whl (44 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 44.6/44.6 kB 1.3 MB/s eta 0:00:00
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (from conformer) (2.1.0+cu121)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch->conformer) (3.13.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch->conformer) (4.9.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch->conformer) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch->conformer) (3.2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch->conformer) (3.1.3)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch->conformer) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch->conformer) (2.1.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch->conformer) (2.1.5)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch->conformer) (1.3.0)
Installing collected packages: einops, conformer
Successfully installed conformer-0.3.2 einops-0.7.0
```

Model

- TransformerEncoderLayer:
  - Base transformer encoder layer in [Attention Is All You Need]()
  - Parameters:
    - d_model: the number of expected features of the input (required).

        - nhead: the number of heads of the multiheadattention models (required).
        - dim_feedforward: the dimension of the feedforward network model (default=2048).
        - dropout: the dropout value (default=0.1).
        - activation: the activation function of intermediate layer, relu or gelu (default=relu).
- TransformerEncoder:
    - TransformerEncoder is a stack of N transformer encoder layers
    - Parameters:
        - encoder_layer: an instance of the TransformerEncoderLayer() class (required).
        - num_layers: the number of sub-encoder-layers in the encoder (required).
        - norm: the layer normalization component (optional).

```
import torch
import torch.nn as nn
import torch.nn.functional as F
# need to install conformer : pip install conformer
from conformer import ConformerBlock


# default : d_model=80, n_spks=600, dropout=0.1, dim_feedforward=256, nhead=2

class Classifier(nn.Module):
    def __init__(self, d_model=256, n_spks=600, dropout=0.1):
        super().__init__()
        # Project the dimension of features from that of input into d_model.
        self.prenet = nn.Linear(40, d_model)
        # TODO:
        #     Change Transformer to Conformer.
        #     https://arxiv.org/abs/2005.08100
        """
        self.encoder_layer = nn.TransformerEncoderLayer(
            d_model=d_model, dim_feedforward=256, nhead=2,
        )
        # self.encoder = nn.TransformerEncoder(self.encoder_layer, num_layers=2)
        """

        # use conformer to replace transformer encoder
        # inner dim of attention = dim_head * head
        # ff_mult = expansion factor for inner linear layer of feedforward
        # conv_expandsion factor : expansion factor for inner linear layer of convolution : dim in con = dim*conv_expandsion factor
        self.conformer_block = ConformerBlock(
            dim = d_model,
            dim_head = 256,
            heads = 1,
            ff_mult = 4,
            conv_expansion_factor = 18,
            conv_kernel_size = 41,
            attn_dropout = dropout,
            ff_dropout = dropout,
            conv_dropout = dropout
        )


        # Project the the dimension of features from d_model into speaker nums.
        self.pred_layer = nn.Sequential(
            #nn.Linear(d_model, d_model),
            nn.ReLU(),
            #nn.BatchNorm1d(d_model),
            #nn.Dropout(0.5),
            nn.Linear(d_model, n_spks),
        )

    def forward(self, mels):
        """
        args:
            mels: (batch size, length, 40)
        return:
            out: (batch size, n_spks)
        """
        # out: (batch size, length, d_model)
        out = self.prenet(mels)
        out = out.permute(1, 0, 2)

        ''' transformer version
        # out: (length, batch size, d_model)
        # permute change dimension order
        out = out.permute(1, 0, 2)
        # The encoder layer expect features in the shape of (length, batch size, d_model) without batch first set to true.
        out = self.encoder_layer(out)
        out = out.transpose(0, 1)
        #out: (batch size, length, d_model)

        '''

        out = self.conformer_block(out)
        out = out.transpose(0, 1)

        # mean pooling
        stats = out.mean(dim=1)
```

```
        # stats: (batch size, d_model)

        # out: (batch, n_spks)
        out = self.pred_layer(stats)
        return out
```

Learning rate schedule

- For transformer architecture, the design of learning rate schedule is different from that of CNN.
- Previous works show that the warmup of learning rate is useful for training models with transformer architectures.
- The warmup schedule
    - Set learning rate to 0 in the beginning.
    - The learning rate increases linearly from 0 to initial learning rate during warmup period.

```
import math

import torch
from torch.optim import Optimizer
from torch.optim.lr_scheduler import LambdaLR


def get_cosine_schedule_with_warmup(
    optimizer: Optimizer,
    num_warmup_steps: int,
    num_training_steps: int,
    num_cycles: float = 0.5,
    last_epoch: int = -1,
):
    """
    Create a schedule with a learning rate that decreases following the values of the cosine function between the
    initial lr set in the optimizer to 0, after a warmup period during which it increases linearly between 0 and the
    initial lr set in the optimizer.

    Args:
        optimizer (:class:`~torch.optim.Optimizer`):
            The optimizer for which to schedule the learning rate.
        num_warmup_steps (:obj:`int`):
            The number of steps for the warmup phase.
        num_training_steps (:obj:`int`):
            The total number of training steps.
        num_cycles (:obj:`float`, `optional`, defaults to 0.5):
            The number of waves in the cosine schedule (the defaults is to just decrease from the max value to 0
            following a half-cosine).
        last_epoch (:obj:`int`, `optional`, defaults to -1):
            The index of the last epoch when resuming training.

    Return:
        :obj:`torch.optim.lr_scheduler.LambdaLR` with the appropriate schedule.
    """

    def lr_lambda(current_step):
        # Warmup
        if current_step < num_warmup_steps:
            return float(current_step) / float(max(1, num_warmup_steps))
        # decadence
        progress = float(current_step - num_warmup_steps) / float(
            max(1, num_training_steps - num_warmup_steps)
        )
        return max(
            0.0, 0.5 * (1.0 + math.cos(math.pi * float(num_cycles) * 2.0 * progress))
        )

    return LambdaLR(optimizer, lr_lambda, last_epoch)
```

Model Function

- Model forward function.

```
import torch


def model_fn(batch, model, criterion, device):
    """Forward a batch through the model."""

    mels, labels = batch
    mels = mels.to(device)
    labels = labels.to(device)

    outs = model(mels)

    loss = criterion(outs, labels)

    # Get the speaker id with highest probability.
    preds = outs.argmax(1)
    # Compute accuracy.
    accuracy = torch.mean((preds == labels).float())

    return loss, accuracy
```

Validate

- Calculate accuracy of the validation set.

```
from tqdm import tqdm
import torch

# record for visualization
valid_loss_record = []
valid_acc_record = []

def valid(dataloader, model, criterion, device):
    """Validate on validation set."""

    model.eval()
    running_loss = 0.0
    running_accuracy = 0.0
    pbar = tqdm(total=len(dataloader.dataset), ncols=0, desc="Valid", unit=" uttr")



    for i, batch in enumerate(dataloader):
        with torch.no_grad():
            loss, accuracy = model_fn(batch, model, criterion, device)
            running_loss += loss.item()
            running_accuracy += accuracy.item()

        pbar.update(dataloader.batch_size)
        pbar.set_postfix(
            loss=f"{running_loss / (i+1):.2f}",
            accuracy=f"{running_accuracy / (i+1):.2f}",
        )

    pbar.close()
    model.train()

    #record for visualization
    valid_loss_record.append(running_loss / len(dataloader))
    valid_acc_record.append(running_accuracy / len(dataloader))

    return running_accuracy / len(dataloader)
```

Main function

```python
from tqdm import tqdm

import torch
import torch.nn as nn
from torch.optim import AdamW
from torch.utils.data import DataLoader, random_split

#record for visualization
train_loss_record = []
train_acc_record = []

def parse_args():
    """arguments"""
    config = {
        "data_dir": "./Dataset",
        "save_path": "model.ckpt",
        "batch_size": 32,
        "n_workers": 24,
        "valid_steps": 2000,
        "warmup_steps": 1000,
        "save_steps": 10000,
        "total_steps": 100000, #default : 70000
    }

    return config


def main(
    data_dir,
    save_path,
    batch_size,
    n_workers,
    valid_steps,
    warmup_steps,
    total_steps,
    save_steps,
):
    """Main function."""
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f"[Info]: Use {device} now!")

    train_loader, valid_loader, speaker_num = get_dataloader(data_dir, batch_size, n_workers)
    train_iterator = iter(train_loader)
    print(f"[Info]: Finish loading data!",flush = True)

    model = Classifier(n_spks=speaker_num).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = AdamW(model.parameters(), lr=1e-3)
    scheduler = get_cosine_schedule_with_warmup(optimizer, warmup_steps, total_steps)
    print(f"[Info]: Finish creating model!",flush = True)

    best_accuracy = -1.0
    best_state_dict = None


    pbar = tqdm(total=valid_steps, ncols=0, desc="Train", unit=" step")

    for step in range(total_steps):
        # Get data
        try:
            batch = next(train_iterator)
        except StopIteration:
            train_iterator = iter(train_loader)
            batch = next(train_iterator)

        loss, accuracy = model_fn(batch, model, criterion, device)
        batch_loss = loss.item()
        batch_accuracy = accuracy.item()

        train_loss_record.append(batch_loss)
        train_acc_record.append(batch_accuracy)

        # Updata model
        loss.backward()
        optimizer.step()
        scheduler.step()
        optimizer.zero_grad()
```

```python
        # Log
        pbar.update()
        pbar.set_postfix(
            loss=f"{batch_loss:.2f}",
            accuracy=f"{batch_accuracy:.2f}",
            step=step + 1,
        )

        # Do validation every valid_step
        if (step + 1) % valid_steps == 0:
            pbar.close()

            valid_accuracy = valid(valid_loader, model, criterion, device)

            # keep the best model
            if valid_accuracy > best_accuracy:
                best_accuracy = valid_accuracy
                best_state_dict = model.state_dict()

            pbar = tqdm(total=valid_steps, ncols=0, desc="Train", unit=" step")

        # Save the best model so far.
        if (step + 1) % save_steps == 0 and best_state_dict is not None:
            torch.save(best_state_dict, save_path)
            pbar.write(f"Step {step + 1}, best model saved. (accuracy={best_accuracy:.4f})")

    pbar.close()


if __name__ == "__main__":
    main(**parse_args())
```

```
[Info]: Use cuda now!
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarni
  warnings.warn(_create_warning_msg(
[Info]: Finish loading data!
[Info]: Finish creating model!
Train: 100% 2000/2000 [01:57<00:00, 17.09 step/s, accuracy=0.19, loss=3.89, step=2000
Valid: 100% 6944/6944 [00:14<00:00, 491.74 uttr/s, accuracy=0.21, loss=3.92]
Train: 100% 2000/2000 [01:36<00:00, 20.64 step/s, accuracy=0.25, loss=3.15, step=4000
Valid: 100% 6944/6944 [00:13<00:00, 523.36 uttr/s, accuracy=0.35, loss=3.01]
Train: 100% 2000/2000 [01:37<00:00, 20.60 step/s, accuracy=0.38, loss=2.84, step=6000
Valid: 100% 6944/6944 [00:13<00:00, 527.09 uttr/s, accuracy=0.45, loss=2.46]
Train: 100% 2000/2000 [01:38<00:00, 20.30 step/s, accuracy=0.56, loss=1.66, step=8000
Valid: 100% 6944/6944 [00:13<00:00, 512.64 uttr/s, accuracy=0.54, loss=2.05]
Train: 100% 2000/2000 [01:41<00:00, 19.68 step/s, accuracy=0.66, loss=1.41, step=1e+4
Valid: 100% 6944/6944 [00:13<00:00, 530.16 uttr/s, accuracy=0.59, loss=1.82]
Train:   0% 2/2000 [00:00<04:59,  6.67 step/s, accuracy=0.44, loss=2.10, step=1e+4]St
Train: 100% 2000/2000 [01:41<00:00, 19.74 step/s, accuracy=0.66, loss=1.20, step=1200
Valid: 100% 6944/6944 [00:13<00:00, 512.17 uttr/s, accuracy=0.63, loss=1.58]
Train: 100% 2000/2000 [01:38<00:00, 20.40 step/s, accuracy=0.72, loss=1.25, step=1400
Valid: 100% 6944/6944 [00:13<00:00, 510.13 uttr/s, accuracy=0.66, loss=1.46]
Train: 100% 2000/2000 [01:40<00:00, 19.90 step/s, accuracy=0.66, loss=1.10, step=1600
Valid: 100% 6944/6944 [00:13<00:00, 512.22 uttr/s, accuracy=0.70, loss=1.28]
Train: 100% 2000/2000 [01:43<00:00, 19.40 step/s, accuracy=0.69, loss=1.17, step=1800
Valid: 100% 6944/6944 [00:13<00:00, 501.63 uttr/s, accuracy=0.71, loss=1.20]
Train: 100% 2000/2000 [01:41<00:00, 19.76 step/s, accuracy=0.81, loss=0.57, step=2e+4
Valid: 100% 6944/6944 [00:13<00:00, 504.55 uttr/s, accuracy=0.74, loss=1.12]
Train:   0% 5/2000 [00:00<01:59, 16.70 step/s, accuracy=0.62, loss=1.13, step=2e+4]St
Train: 100% 2000/2000 [01:41<00:00, 19.70 step/s, accuracy=0.88, loss=0.55, step=2200
Valid: 100% 6944/6944 [00:14<00:00, 488.62 uttr/s, accuracy=0.75, loss=1.09]
Train: 100% 2000/2000 [01:43<00:00, 19.40 step/s, accuracy=0.75, loss=1.16, step=2400
Valid: 100% 6944/6944 [00:13<00:00, 500.16 uttr/s, accuracy=0.75, loss=1.06]
Train: 100% 2000/2000 [01:44<00:00, 19.22 step/s, accuracy=0.84, loss=0.49, step=2600
Valid: 100% 6944/6944 [00:14<00:00, 492.78 uttr/s, accuracy=0.77, loss=0.98]
Train: 100% 2000/2000 [01:41<00:00, 19.72 step/s, accuracy=0.84, loss=0.69, step=2800
Valid: 100% 6944/6944 [00:15<00:00, 458.72 uttr/s, accuracy=0.77, loss=1.00]
Train: 100% 2000/2000 [01:43<00:00, 19.36 step/s, accuracy=0.94, loss=0.54, step=3e+4
Valid: 100% 6944/6944 [00:14<00:00, 477.80 uttr/s, accuracy=0.79, loss=0.92]
Train:   0% 6/2000 [00:00<01:29, 22.32 step/s, accuracy=0.88, loss=0.46, step=3e+4]St
Train: 100% 2000/2000 [01:43<00:00, 19.24 step/s, accuracy=0.88, loss=0.56, step=3200
Valid: 100% 6944/6944 [00:15<00:00, 460.40 uttr/s, accuracy=0.78, loss=0.94]
Train: 100% 2000/2000 [01:45<00:00, 18.88 step/s, accuracy=0.84, loss=0.57, step=3400
Valid: 100% 6944/6944 [00:15<00:00, 448.46 uttr/s, accuracy=0.80, loss=0.87]
Train: 100% 2000/2000 [01:45<00:00, 18.90 step/s, accuracy=0.91, loss=0.32, step=3600
Valid: 100% 6944/6944 [00:15<00:00, 437.88 uttr/s, accuracy=0.81, loss=0.84]
Train: 100% 2000/2000 [01:45<00:00, 19.00 step/s, accuracy=0.94, loss=0.19, step=3800
Valid: 100% 6944/6944 [00:15<00:00, 454.58 uttr/s, accuracy=0.81, loss=0.82]
Train: 100% 2000/2000 [01:44<00:00, 19.17 step/s, accuracy=0.84, loss=0.44, step=4e+4
Valid: 100% 6944/6944 [00:14<00:00, 465.75 uttr/s, accuracy=0.81, loss=0.80]
Train:   0% 6/2000 [00:00<01:32, 21.62 step/s, accuracy=0.84, loss=0.43, step=4e+4]St
Train: 100% 2000/2000 [01:44<00:00, 19.15 step/s, accuracy=1.00, loss=0.12, step=4200
Valid: 100% 6944/6944 [00:16<00:00, 411.75 uttr/s, accuracy=0.83, loss=0.74]
Train: 100% 2000/2000 [01:44<00:00, 19.22 step/s, accuracy=0.94, loss=0.20, step=4400
Valid: 100% 6944/6944 [00:16<00:00, 412.12 uttr/s, accuracy=0.83, loss=0.76]
Train: 100% 2000/2000 [01:45<00:00, 19.02 step/s, accuracy=0.88, loss=0.25, step=4600
Valid: 100% 6944/6944 [00:16<00:00, 414.40 uttr/s, accuracy=0.83, loss=0.75]
Train: 100% 2000/2000 [01:45<00:00, 19.03 step/s, accuracy=0.91, loss=0.23, step=4800
```

Inference

Dataset of inference

```
import os
import json
import torch
from pathlib import Path
from torch.utils.data import Dataset


class InferenceDataset(Dataset):
    def __init__(self, data_dir):
        testdata_path = Path(data_dir) / "testdata.json"
        metadata = json.load(testdata_path.open())
        self.data_dir = data_dir
        self.data = metadata["utterances"]

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        utterance = self.data[index]
        feat_path = utterance["feature_path"]
        mel = torch.load(os.path.join(self.data_dir, feat_path))

        return feat_path, mel


def inference_collate_batch(batch):
    """Collate a batch of data."""
    feat_paths, mels = zip(*batch)

    return feat_paths, torch.stack(mels)
```

Main funcrion of Inference

```
import json
import csv
from pathlib import Path
#from tqdm.notebook import tqdm
from tqdm import tqdm as tqdm


import torch
from torch.utils.data import DataLoader

def parse_args():
    """arguments"""
    config = {
        "data_dir": "./Dataset",
        "model_path": "./model.ckpt",
        "output_path": "./output.csv",
    }

    return config


def main(
    data_dir,
    model_path,
    output_path,
):
    """Main function."""
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f"[Info]: Use {device} now!")

    mapping_path = Path(data_dir) / "mapping.json"
    mapping = json.load(mapping_path.open())
```