# VIVOS recipe

ESPNet

# Tutorial

https://github.com/espnet/notebook/blob/master/espnet2_tutorial_2021_CMU_11751_18781.ipynb

# ESPNET recipe examples

ESPnet has a number of recipes (73 recipes on Sep. 16, 2021). Let's first check https://github.com/espnet/espnet/blob/master/egs2/README.md

Please also check the general usage of the recipe in https://espnet.github.io/espnet/espnet2_tutorial.html#recipes-using-espnet2

# ESPnet2 demonstration

(jupyter-lab, run as admin, fit_caohoc_espnet2_streaming_asr_demo.ipynb)

# Full installation

**For Docker:** https://github.com/espnet/espnet/blob/master/docker/README.md
**Installation of required tools:** https://espnet.github.io/espnet/installation.html#requirements
**Download espnet:**

```
git clone --depth 5 https://github.com/espnet/espnet
```

**Setup Python environment based on anaconda**

```
%cd /content/espnet/tools
./setup_anaconda.sh anaconda espnet 3.8
```

**Install espnet, This includes the installation of PyTorch and other tools, We just specify CUDA_VERSION=10.2 for the latest PyTorch (1.9.0)**

```
%cd /content/espnet/tools
make CUDA_VERSION=10.2|
```

**Install NIST SCTK toolkit for scoring**

```
%cd /content/espnet/tools
./installers/install_sctk.sh
```

**Check installation**

```
%cd /content/espnet/tools
. ./activate_python.sh; python3 check_install.py
```

# VIVOS recipe

```
% cd espnet/egs2/vivos/asr1

egs2/vivos/asr1/
 - conf/      # Configuration files for training, inference, etc.
 - scripts/   # Bash utilities of espnet2
 - pyscripts/ # Python utilities of espnet2
 - steps/     # From Kaldi utilities
 - utils/     # From Kaldi utilities
 - db.sh      # The directory path of each corpora
 - path.sh    # Setup script for environment variables
 - cmd.sh     # Configuration for your backend of job scheduler
 - run.sh     # Entry point
 - asr.sh     # Invoked by run.sh
```

# run.sh

**run.sh** can call **asr.sh**, which completes the entire speech recognition experiments, including **data preparation**, **training**, **inference**, and **scoring**. They are based on separate stages (totally 15 stages).

```
…

train_set="train_nodev"
valid_set="train_dev"
test_sets="train_dev test"
asr_config=conf/train_asr.yaml
inference_config=conf/decode.yaml
lm_config=conf/train_lm_char.yaml
use_lm=true
use_wordlm=false
word_vocab_size=7184

./asr.sh                                    \
    --lang vi                               \
    --audio_format wav                      \
    --feats_type raw                        \
…
    --train_set "${train_set}"              \
    --valid_set "${valid_set}"              \
    --test_sets "${test_sets}"              \
    --lm_train_text "data/${train_set}/text" "$@"
```

# Data preparation

Stage 1: Data preparation for training, validation, and evaluation data

Note that --stage <N> is to start the stage and --stop_stage <N> is to stop the stage.

```
./run.sh --stage 1 --stop_stage 1
```

Check training, validation, and test data in espnet/egs2/vivos/asr1/data

```
test
train
train_dev
train_nodev
```

# Kaldi format

https://kaldi-asr.org/doc/data_prep.html

ls -1 espnet/egs2/vivos/asr1/data/train_nodev/

spk2utt # Speaker information

text  # Transcription file

utt2spk # Speaker information

wav.scp # Audio file

# Stage 2: Speed Perturbation (Data Augmentation)

We do not use speed perturbation for this demo. But you can turn it on by adding an argument **--speed_perturb_factors "0.9 1.0 1.1"** to the shell script

./run.sh --stage 2 --stop_stage 2

# Stage 3: Format wav.scp: data/ -> dump/raw

We dump the data with specified format for the efficient use of the data.

Note that **--nj <N>** means the number of CPU jobs. Please set it appropriately by considering your CPU resources and disk access. (Default nj=32)

```
./run.sh --stage 3 --stop_stage 3
```

# Stage 4: Remove long/short data dump/raw/org -> dump/raw

There are too long and too short audio data, which are harmful for our efficient training. Those data are removed from the list.

```
./run.sh --stage 4 --stop_stage 4
```

## Stage 5: Generate token_list from dump/raw/train_nodev/text using BPE

We make a dictionary based on the English character in this example.

```
./run.sh --stage 5 --stop_stage 5
```

Let's check the content of the dictionary. There are several special symbols, e.g.,

```
<blank>   used for CTC
<unk>     unknown symbols do not appear in the training data
<sos/eos> start and end sentence symbols
```

```
cat data/vi_token_list/char/tokens.txt
```

# Language Modeling

If skip training LM (stages 6 - 9), in run.sh

```
    use_lm=false
else
    ./run.sh --stage 6 --stop_stage 9
```

# End-to-end ASR

Stage 10: ASR collect stats

```
./run.sh --stage 10 --stop_stage 10
```

Stage 11: ASR Training

./run.sh --stage 11 --stop_stage 11 **--ngpu 1**

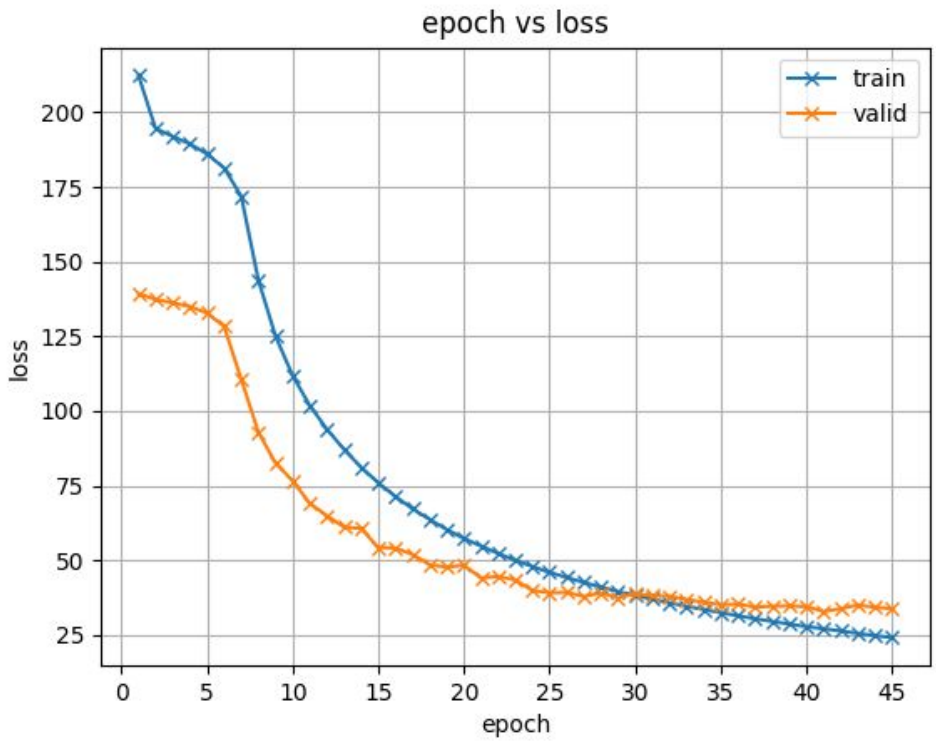
Stage 12: Decoding

```
./run.sh --stage 12 --stop_stage 12 \
        --inference_asr_model valid.loss.ave.pth \
        --inference_nj 4 --use_lm false
```

Stage 13: Scoring

```
./run.sh --stage 13 --stop_stage 13 --use_lm false
```

# Monitor the training process




espnet/egs2/vivos/asr1/exp/asr_train_asr_raw_vi_char/images/loss.png

# Monitor the training process

Full log file espnet/egs2/vivos/asr1/exp/asr_train_asr_raw_vi_char/train.log

# Change the training configs

espnet/egs2/vivos/asr1/conf/train_asr.yaml

Other configs:

- LSTM-based E2E ASR

espnet/egs2/an4/asr1/conf/train_asr_rnn.yaml

- Transformer based E2E ASR
espnet/egs2/an4/asr1/conf/train_asr_transformer.yaml

# References

Luong Hieu Thi, An end-to-end Vietnamese speech recognition recipe using ESPnet toolkit, https://www.hieuthi.com/blog/2019/10/22/end-to-end-vietnamese-speech-recognition-espnet.html

VLSP 2021 - Technical reports - ASR Task

https://drive.google.com/drive/folders/1uADWIoXbPdHLfmjdLuH7VlT98LIpPwu4?usp=sharing