Finetuning ASR

Pietro De Angeli

16/9/2024

Contents

In the section we will explore transfer learning, with a focus on fine-tuning an ASR model. This need arises when the transcription models used do not meet the specific characteristics of the audio to be processed. A common example is a *general-purpose* model that needs to be adapted to a specific domain, such as recognizing terminology from a specialized vocabulary or a specific language. Another case is transcribing speech with a strong accent: even within countries that share the same language, pronunciation and vocabulary usage can vary significantly. The goal of this project is to test the feasibility of such operations using limited resources and open-source models from the NeMo framework, highlighting the main challenges and optimal pipeline for addressing this task.

This section of the thesis explores two solutions to improve the transcription accuracy of an audio model by adapting it to a new language. The first solution involves fine-tuning the decoder of the ASR model, adapting it to the new language without modifying the encoder. In this approach, the encoder remains unchanged and continues to map audio features into a language-neutral representation. The fine-tuning focuses on the decoder, training it to interpret those representations and convert them correctly into the target language. This approach allows optimizing the model for the new language efficiently in terms of resources and time.

The second solution explored is training an N-gram model to improve transcription by modeling the probabilities of words characteristic of the target language or domain. This statistical model helps reduce transcription errors by improving the coherence of transcribed words based on their frequency and occurrence patterns in a specific linguistic context.

1 Decoder Fine-tuning

In this section, the decoder is fine-tuned using audio files in the new language. The encoder is kept frozen, as the acoustic features are common across Italian, English, and most languages.

1.1 Dataset

The dataset used is from Mozilla Common Voice (MCV), a platform providing open-source speech datasets in various languages. We used version 6.1 ($\tilde{5}$GB). While larger datasets exist, a smaller version was chosen for illustrative purposes.

1.2 Dataset Preprocessing

The original dataset consists of .mp3 files and a .tsv file with transcription labels. However, NeMo recommends using MCV APIs to download preprocessed .wav files organized into three splits (`train`, `validation`, `test`) with manifests available at runtime. This solution requires integration with HuggingFace and MCV APIs.

To support generic datasets, we developed a custom preprocessing pipeline. This includes: converting .mp3 files to .wav, resampling to 16 kHz (standard for NeMo models), and converting audio to mono channel.

The processed dataset must contain three manifests: `train`, `validation`, and `test`. Each manifest includes `text`, `audio_filepath`, and `duration`. Our script can automatically convert files and compute durations.

The manifests are split approximately into 80% for training, 10% for validation, and 10% for testing. To simplify training and reduce computational load, the training set was reduced to 1 hour (100 clips), and validation/test sets to 10 minutes (15 clips).

Since we are fine-tuning on Italian, the model vocabulary had to be adapted to include accented characters absent in the default English vocabulary. This required extracting all characters from the `text` field, excluding punctuation and uppercase letters, ensuring the vocabulary only contains necessary characters.

Additionally, data augmentation was applied during training by injecting noise at runtime to improve robustness.

1.3 Model Selection

We used `stt_en_quartznet15x5`, a lightweight model with strong performance on small datasets thanks to its CNN-based architecture. Unlike Conformer (used in diarization), which includes transformers for contextual modeling, QuartzNet only uses CNNs.

`stt_en_quartznet15x5` was trained on 7000 hours of English audio, handling durations from 0.1s to 16s and beyond. It does not support punctuation or capitalization, which is acceptable for our goals.

1.4 Hyperparameters

NVIDIA researchers showed that fine-tuning yields better results than training from scratch. We adopted hyperparameters from:

Listing 1: Hyperparameters settings

```
with open_dict(cfg.optim):
```

```
cfg.optim.lr = 0.01
cfg.optim.weight_decay = 0.001
cfg.optim.sched.warmup_ratio = 0.05
cfg.optim.sched.min_lr = 1e-5
```

1.5 Project Structure

In the **dataset** folder, datasets used for fine-tuning are stored in subfolders named `<dataset_name>-<lang_id>`, each containing `train`, `validation`, `test` subfolders and a `clips` folder with .wav files.

The **models** folder contains the saved *quartznet_en_15x5* model in .nemo format for NeMo compatibility. Models can be loaded using `.from_pretrained()` and saved after training.

The **utils** folder contains:

- `preprocess_dataset.py`: handles preprocessing steps like removing invalid files, cleaning characters, generating NeMo-compatible manifests.
  - `convert_mp3_to_wav`: converts audio to .wav, resamples to 16kHz, sets mono channel.
  - `process_tsv`: parses .tsv and outputs .csv with audio paths and transcriptions.
  - `delete_invalidated`: deletes audio files listed in invalidation .tsv.
  - `keep_first_n_lines`: keeps first `n` lines of a file, useful for testing.
  - `MVCtoNEMO`: converts .csv manifests to NeMo-compatible .json format, computes duration.
- `train.py`: manages fine-tuning using manifest files, configures model hyperparameters, freezes encoder, calls `Trainer.fit()`, saves the model.

1.6 Results

Although the model was trained for only 5 epochs on limited data, it was able to produce some predictions:

Listing 2: Fine-tuning results

```
REF:appese le scarpette al chiodo assume il ruolo di
    allenatore dei portieri
```

```
PRE:a pese le scampeta    ciodo  assume   rolo  di
    allenatar  di  portiri
```

The transcription quality is not optimal, but promising considering the minimal training data. NVIDIA's team fine-tuned the same model using the full dataset and over 100 epochs to obtain `stt_it_quartznet_15x5`.

2 N-gram

This section explores hypothesis rescoring using N-gram models. The **quartznet_it_15x5** model (fine-tuned on 1000h of Italian) was used instead of the English model to compute sequence probabilities for scoring.

2.1 Dataset

The N-gram model was trained using two datasets: general Italian (from MCV train split) and domain-specific data (TV transcripts provided by the hosting company). The domain dataset was 100KB and the general 3.5MB. Both were cleaned to include only characters present in the Italian vocabulary. Using KenLM, a weight of 70% was assigned to the general dataset and 30% to the domain dataset.

2.2 Language Model

The interpolated 4-gram model was built using KenLM. The final LM is exported in .arpa and binary format (.bin) using: `build_binary model.arpa model.bin`

A lexicon file (.lexicon) was also created to define valid words.

2.3 Model Integration

To apply the N-gram, decoder settings were updated with:

- **Decoding strategy**: `flashlight`, for efficient beam search
- **Alpha, Beta**: control language model weight and length penalty
- **Beam threshold**: pruning unlikely paths
- **Unknown weight**: penalizes unknown words

2.4 Results

The N-gram rescoring showed modest improvements. QuartzNet occasionally produces phonetically similar but incorrect words, so minor character errors result in WER penalties. Tests used the same audio and WER calculation as in Chapter 3.5.

Table 1: WER N-gram

Model	**WER (%)**	**Insertions**	**Deletions**	**Substitutions**
Original model	45.62	455	2070	5658
N-gram rescoring	**40.73**	**264**	**2426**	**4869**

WER improved by about 5%, mainly reducing insertions and substitutions.

Table 2: Italian WER N-gram

Model	**WER (%)**	**Insertions**	**Deletions**	**Substitutions**
NVIDIA quartznet	45.62	455	2070	5658
N-gram rescoring	**40.73**	**264**	**2426**	**4869**