Fine-Tuning week of XLSR-Wav2Vec2 on 60 languages

Welcome to the fine-tuning week! The goal of this week is to have state-of-the-art automatic speech recognition (ASR) models in as many languages as possible. The fine-tuning week ends on Friday, the 26th March at midnight PST time.

Participants are encouraged to fine-tune the pretrained facebook/wav2vec2-large-xlsr-53 checkpoint on one or more of the 60 languages of Common Voice dataset. Furthermore, it is very much appreciated if participants fine-tune XLSR-Wav2Vec2 on a language that is not included in the Common Voice dataset.

All fine-tuned models uploaded until Friday, the 26th March midnight PST, will be taken into account for competition, and the best model per language will be awarded a prize if the best model performs reasonably well. The testing data to evaluate the models will be the official Common Voice dataset `test data` of version 6.1. Again, participants are very much encouraged to fine-tune XLSR-Wav2Vec2 on languages that are not found in the Common Voice dataset since those languages are even more likely to be underrepresented in the speech community. Each model fine-tuned on a language not found in Common Voice, will be evaluated by the Hugging Face team after Friday, the 26th March at midnight PST, and if the model performs reasonably well, the model receives a prize as well. For more information on which data can be used for training, how the models are evaluated exactly, and what type of data preprocessing can be used, please see "Training and Evaluation Rules".

Please keep in mind: The spirit of the fine-tuning week is to provide state-of-the-art speech recognition in as many languages as possible to the community! So while we encourage healthy competition between people/groups of the same language so that better results are obtained, it is extremely important that we help each other and share our insights with the whole team/community. What matters in the end is what has been achieved by the team as a whole during the fine-tuning week. That being said, we strongly encourage people to share tips & tricks on the forum or Slack, help each other when team members encounter bugs, and work in groups. To make it easier to share and help, forum threads have been created under the name {language} ASR: Fine-Tuning Wav2Vec2, e.g. here. It is very much possible that prizes will be given to groups of people instead of individuals. Also, don't hesitate to ask questions, propose improvements to the organization, to the material given to participants, etc...

Table of Contents

Organization of the fine tuning week

The week officially starts on 22.03.2021 and ends on 29.03.2021, but you are more than welcome to start fine-tuning models before the start date. General questions you might have, general problems you encounter, and general tips can be shared directly on the Slack channel (see this post on how to be added to Slack). More language-specific questions or specific bugs should be posted on the forum (feel free to use already existing language-specific threads, *e.g.* this one or open a new one if there is no thread for your language yet) or directly on github if you think some code or document needs correction/improvement. Starting on Monday, the 22.03.2021, the Hugging Face team will try to provide an overview of currently trained models along with their evaluation results. All the necessary information on:

- How to fine-tune the XLSR model
- How to upload the model
- How to share your evaluation results & training/eval script
- What are the training/evaluation rules

can be found in the sections below. If something is still unclear, feel free to drop a message in the Slack channel.

How to fine tune XLSR Wav2Vec2

This chapter gives an in-detail explanation of how to fine-tune Facebook's multi-lingual Wav2vec2 on any language of the Common Voice dataset.

Two possible setups can be used to fine-tune Wav2Vec2. The easiest setup is to simply use google colab. It is possible to train the full model in a *free* google colab, but it is recommended to use google colab pro since it is more stable.

The other option is to run a script locally. While this can be more difficult to set up, it also means that you have more control over the training run and probably access to better GPUs than you would have in a google colab. For small datasets, it is usually totally sufficient to train your model in a google colab. For larger and thus more memory-intensive datasets, it is probably better to fine-tune the model locally.

For each option, we explain in detail how to fine-tune XLSR-Wav2Vec2 in the following.

Google colab setup

Note: Instead of reading the following section, you can simply watch this video, where Patrick explains how to adapt the google colab for your specific language.

1.: If you plan on training XLSR-Wav2Vec2 in a google colab, you should first make sure to have a valid gmail account. You can sign up for a gmail account here. Having successfully signed up for gmail, you can now sign in to your account to make sure you are logged in when opening new tabs in your browser.

2.: Next, head over to the official Fine-Tune XLSR-Wav2Vec2 with 🤗 Transformes google colab. The first thing you should do is to make a copy of it - click `->File->Save a copy in Drive`. This should save a copy of the google colab in your google drive.

3.: Now it is highly recommended to carefully read the google colab without running the cells yet. You should get an understanding of the model is trained and what you will have to change when training the model in a different language. Having done so, you can again head over to Common Voice and pick a language you want to fine-tune facebook/wav2vec2-large-xlsr-53 on. Make sure you remember the language code (For each language, you can find it under the field "*Version*". It corresponds to **all characters before the first underscore**. *E.g.* for Greek it is *el*, while for Irish it is *ga-IE*.

4.: Now you should replace the language code used for the demo of this colab, being *tr* for Turkish with the language code corresponding to the language you just chose in the **second** cell of the google colab. This will load the correct data for your language.

5.: It is time to start running the google colab! Make sure that you have selected "GPU" as your runtime environment and you can start running the cells one-by-one. Make sure you attentively read the text between the cells to understand what is happening and to eventually correct the cells to improve the fine-tuning script for your language. Things you might want to improve/change:

- Data loading. It is very much recommended to use more than just the official training data of the Common Voice dataset. If you find more data on the internet, feel free to use it! Check out the section "How to combined multiple datasets into one"
- Data Processing. You should adapt the data processing to your specific language. In data processing, you should make the data more uniform so that it will be easier for the model to learn how to classify speech in your data. Here it can be really helpful to be proficient in the language to know what can be done to simplify the language without changing the meaning. Data processing methods include, but are not limited to:
    - Normalizing your data. Make sure all characters are lower-cased.
    - Remove typographical symbols and punctuation marks. See a list here. Be careful to not remove punctuation marks that can change the meaning of the sentence. *E.g.* you should not remove the single quotation mark `'` in English, as it would change the words `"it's"` to `"its"` which is a different word and has thus a different meaning. For more tips on data processing see "How to effectively preprocess the data"
- Hyperparameter Tuning. Depending on the size of the data you should probably change the hyperparameters of the google colab. You can change any parameter you like. For more tips and tricks see "How to do hyperparameter tuning for my language"

When running the google colab make sure that you uncomment the cell corresponding to mounting your google drive to the colab. This cell looks as follows:

```
# from google.colab import drive
# drive.mount('/content/gdrive/')
```

Uncomment it, run it, and follow the instructions to mount your google drive. This way you can be sure that the model parameters and created tokenizer & feature extractor files are saved in **your** google drive.

Also, make sure that you uncomment the cells corresponding to save the preprocessing files and trained model weights to your drive. Otherwise, you might lose a trained model if you google crashes. You should change the name of your model from `wav2vec2-large-xlsr-turkish-demo` to `wav2vec2-large-xlsr-{your_favorite_name}`.

Those cells correspond to:

```
# processor.save_pretrained("/content/gdrive/MyDrive/wav2vec2-large-xlsr-turkish-demo")
```

and the line:

```
  output_dir="/content/gdrive/MyDrive/wav2vec2-large-xlsr-turkish-demo",
```

further below (which should already be uncommented).

Having finished the training you should find the following files/folders under the folder `wav2vec2-large-xlsr-{your_favorite_name}` in your google drive:

- `preprocessor_config.json` - the parameters of the feature extractor
- `special_tokens_map.json` - the special token map of the tokenizer
- `tokenizer_config.json` - the parameters of the tokenizer
- `vocab.json` - the vocabulary of the tokenizer
- `checkpoint-{...}/` - the saved checkpoints saved during training. Each checkpoint should contain the files: `config.json`, `optimizer.pt`, `pytorch_model.bin`, `scheduler.pt`, `training_args.bin`. The files `config.json` and `pytorch_model.bin` define your model.

If you are happy with your training results it is time to upload your model! Download the following files to your local computer: **`preprocessor_config.json`**, **`special_tokens_map.json`**, **`tokenizer_config.json`**, **`vocab.json`**, **`config.json`**, **`pytorch_model.bin`**. Those files fully define a XLSR-Wav2Vec2 model checkpoint.

Awesome you have successfully trained a XLSR-Wav2Vec2 model 😎. Now you can jump to the section "How to upload my trained checkpoint"

Local machine

We have provided `run_common_voice.py` script to run fine-tuning on local machine. The script is similar to the colab but allows you to launch training using command line, save and continue training from previous checkpoints and launch training on multiple GPUs. For bigger datasets, we recommend to train Wav2Vec2 locally instead of in a google colab.

1. To begin with, we should clone transformers localy and install all the required packages.

First, you need to clone the `transformers` repo with:

```
$ git clone https://github.com/huggingface/transformers.git
```

Second, head over to the `examples/research_projects/wav2vec2` directory, where the `run_common_voice.py` script is located.

```
$ cd transformers/examples/research_projects/wav2vec2
```

Third, install the required packages. The packages are listed in the `requirements.txt` file and can be installed with

```
$ pip install -r requirements.txt
```

```
**Note**: Installing the latest version of `torchaudio` will also upgrade `torch` to
it's latest stable version. If you are using specific version of `torch` then make
sure
to use the correct `torchaudio` version compatible with your version of `torch`. By
default the `requirements.txt` will install the latest version of `torchaudio`.
```

2. Next, take a look at the `run_common_voice.py` script to get an understanding of how it works. In short the script does the following:

    - Load the given common voice dataset
    - Create vocab for the language
    - Load the model with given hyperparameters
    - Pre-process the dataset to input into the model
    - Run training
    - Run evaluation

3. The following examples show how you can launch fine-tuning for the common voice dataset. Here we will run the script on the *Turkish* Common Voice dataset for demonstration purposes.

    To lanuch fine-tuninig on a single GPU:

```
python run_common_voice.py \
    --model_name_or_path="facebook/wav2vec2-large-xlsr-53" \
    --dataset_config_name="tr" \ # use this argument to specify the language
code
    --output_dir=./wav2vec2-large-xlsr-turkish-demo \
    --overwrite_output_dir \
    --num_train_epochs="5" \
    --per_device_train_batch_size="16" \
    --learning_rate="3e-4" \
    --warmup_steps="500" \
    --evaluation_strategy="steps" \
    --save_steps="400" \
    --eval_steps="400" \
    --logging_steps="400" \
    --save_total_limit="3" \
    --freeze_feature_extractor \
    --feat_proj_dropout="0.0" \
    --layerdrop="0.1" \
    --gradient_checkpointing \
    --fp16 \
    --group_by_length \
    --do_train --do_eval
```

    To lanuch fine-tuninig on multiple GPUs:

```
python -m torch.distributed.launch \
    --nproc_per_node 4 run_common_voice.py \
    --model_name_or_path="facebook/wav2vec2-large-xlsr-53" \
    --dataset_config_name="tr" \ # use this argument to specify the language
code
    --output_dir=./wav2vec2-large-xlsr-turkish-demo \
    --overwrite_output_dir \
    --num_train_epochs="5" \
    --per_device_train_batch_size="16" \
    --learning_rate="3e-4" \
    --warmup_steps="500" \
    --evaluation_strategy="steps" \
```

```
    --save_steps="400" \
    --eval_steps="400" \
    --logging_steps="400" \
    --save_total_limit="3" \
    --freeze_feature_extractor \
    --feat_proj_dropout="0.0" \
    --layerdrop="0.1" \
    --gradient_checkpointing \
    --fp16 \
    --group_by_length \
    --do_train --do_eval
```

The above command will launch the training on 4 GPUs. Use the `--nproc_per_node` option to specify the number of GPUs.

Once the training is finished, the model and checkpoints will be saved under the directory specified by the `--output_dir` argument.

4. The script also allows you to resume training from the last saved checkpoint. To resume training from last saved checkpoint remove the `--overwrite_output_dir` option and run the same command again. And to continue training from a specific checkpoint, keep the `--overwrite_output_dir` option and pass the path of the checkpoint as `--model_name_or_path`.

As the script is based on the `Trainer` API, refer to the Trainer docs for more information about `Trainer` and `TrainingArguments`.

OVH cloud has generously offered free compute for this sprint. Please refer to this video to get started with OVH.

How to upload my trained checkpoint

To upload your trained checkpoint, you have to create a new model repository on the 🤗 model hub, from this page: https://huggingface.co/new

> *You can also follow the more in-depth instructions here if needed.*

Having created your model repository on the hub, you should clone it locally:

```
git lfs install

git clone https://huggingface.co/username/your-model-name
```

Then and add the following files that fully define a XLSR-Wav2Vec2 checkpoint into the repository. You should have added the following files.

- `preprocessor_config.json`
- `special_tokens_map.json`
- `tokenizer_config.json`
- `vocab.json`
- `config.json`
- `pytorch_model.bin`

Having added the above files, you should run the following to push files to your model repository.

```
git add . && git commit -m "Add model files" && git push
```

The next **very important** step is to create the model card. For people to use your fine-tuned model it is important to understand:

- What kind of model is it?
- What is your model useful for?
- What data was your model trained on?
- How well does your model perform?

All these questions should be answered in a model card which is the first thing people see when visiting your model on the hub under `https://huggingface.co/{your_username}/{your_modelname}`.

Note: It is extremely important that you add this model card or else we cannot find your model and thus cannot take the model into account for the final evaluation.

How to create the readme

The model card is written in markdown (`.md`) and should be added by simply clicking on the "Add model card" button which is found on the top right corner. You are encouraged to copy-paste the following template into your model card.

Make sure that instead of copying the output of the markdown file you copy the **raw** version of the following part.

To get the raw version of this file, simply click on the " `raw` " button on the top right corner of this file next to " `blame` " and copy everything below the marker. Make sure that you read and consequently remove all #TODO: statements from the model card.

<==========Copy raw version from here==========

language: {lang_id} #TODO: replace {lang_id} in your language code here. Make sure the code is one of the *ISO codes* of this site. datasets:

- common_voice #TODO: remove if you did not use the common voice dataset
- TODO: add more datasets if you have used additional datasets. Make sure to use the exact same dataset name as the one found here. If the dataset can not be found in the official datasets, just give it a new name metrics:
- wer tags:
- audio
- automatic-speech-recognition
- speech
- xlsr-fine-tuning-week license: apache-2.0 model-index:
- name: {human_readable_name} #TODO: replace {human_readable_name} with a name of your model as it should appear on the leaderboard. It could be something like `Elgeish XLSR Wav2Vec2 Large 53` results:
    - task: name: Speech Recognition type: automatic-speech-recognition dataset: name: Common Voice {lang_id} #TODO: replace {lang_id} in your language code here. Make sure the code is one of the *ISO codes* of this site. type: common_voice args: {lang_id} #TODO: replace {lang_id} in your language code here. Make sure the code is one of the *ISO codes* of this site. metrics:
        - name: Test WER type: wer value: {wer_result_on_test} #TODO (IMPORTANT): replace {wer_result_on_test} with the WER error rate you achieved on the common_voice test set.

It should be in the format XX.XX (don't add the % sign here). **Please** remember to fill out this value after you evaluated your model, so that your model appears on the leaderboard. If you fill out this model card before evaluating your model, please remember to edit the model card afterward to fill in your value

---

Wav2Vec2-Large-XLSR-53-{language} #TODO: replace language with your {language}, *e.g.* French

Fine-tuned facebook/wav2vec2-large-xlsr-53 on {language} using the Common Voice, ... and ... dataset{s}. #TODO: replace {language} with your language, *e.g.* French and eventually add more datasets that were used and eventually remove common voice if model was not trained on common voice When using this model, make sure that your speech input is sampled at 16kHz.

Usage

The model can be used directly (without a language model) as follows:

```
import torch
import torchaudio
from datasets import load_dataset
from transformers import Wav2Vec2ForCTC, Wav2Vec2Processor

test_dataset = load_dataset("common_voice", "{lang_id}", split="test[:2%]") #TODO:
replace {lang_id} in your language code here. Make sure the code is one of the *ISO
codes* of [this](https://huggingface.co/languages) site.

processor = Wav2Vec2Processor.from_pretrained("{model_id}") #TODO: replace
{model_id} with your model id. The model id consists of
{your_username}/{your_modelname}, *e.g.* `elgeish/wav2vec2-large-xlsr-53-arabic`
model = Wav2Vec2ForCTC.from_pretrained("{model_id}") #TODO: replace {model_id} with
your model id. The model id consists of {your_username}/{your_modelname}, *e.g.*
`elgeish/wav2vec2-large-xlsr-53-arabic`

resampler = torchaudio.transforms.Resample(48_000, 16_000)

# Preprocessing the datasets.
# We need to read the aduio files as arrays
def speech_file_to_array_fn(batch):
    speech_array, sampling_rate = torchaudio.load(batch["path"])
    batch["speech"] = resampler(speech_array).squeeze().numpy()
    return batch

test_dataset = test_dataset.map(speech_file_to_array_fn)
inputs = processor(test_dataset[:2]["speech"], sampling_rate=16_000,
return_tensors="pt", padding=True)

with torch.no_grad():
    logits = model(inputs.input_values, attention_mask=inputs.attention_mask).logits
```

```
predicted_ids = torch.argmax(logits, dim=-1)

print("Prediction:", processor.batch_decode(predicted_ids))
print("Reference:", test_dataset[:2]["sentence"])
```

Evaluation

The model can be evaluated as follows on the {language} test data of Common Voice. # TODO: replace #TODO: replace language with your {language}, *e.g.* French

```
import torch
import torchaudio
from datasets import load_dataset, load_metric
from transformers import Wav2Vec2ForCTC, Wav2Vec2Processor
import re

test_dataset = load_dataset("common_voice", "{lang_id}", split="test") #TODO:
replace {lang_id} in your language code here. Make sure the code is one of the *ISO
codes* of [this](https://huggingface.co/languages) site.
wer = load_metric("wer")

processor = Wav2Vec2Processor.from_pretrained("{model_id}") #TODO: replace
{model_id} with your model id. The model id consists of
{your_username}/{your_modelname}, *e.g.* `elgeish/wav2vec2-large-xlsr-53-arabic`
model = Wav2Vec2ForCTC.from_pretrained("{model_id}") #TODO: replace {model_id} with
your model id. The model id consists of {your_username}/{your_modelname}, *e.g.*
`elgeish/wav2vec2-large-xlsr-53-arabic`
model.to("cuda")

chars_to_ignore_regex = '[\,\?\.\!\-\;\:\"\“]'  # TODO: adapt this list to include
all special characters you removed from the data
resampler = torchaudio.transforms.Resample(48_000, 16_000)

# Preprocessing the datasets.
# We need to read the aduio files as arrays
def speech_file_to_array_fn(batch):
    batch["sentence"] = re.sub(chars_to_ignore_regex, '', batch["sentence"]).lower()
    speech_array, sampling_rate = torchaudio.load(batch["path"])
    batch["speech"] = resampler(speech_array).squeeze().numpy()
    return batch

test_dataset = test_dataset.map(speech_file_to_array_fn)

# Preprocessing the datasets.
# We need to read the aduio files as arrays
def evaluate(batch):
    inputs = processor(batch["speech"], sampling_rate=16_000, return_tensors="pt",
padding=True)

    with torch.no_grad():
```

```
        logits = model(inputs.input_values.to("cuda"),
attention_mask=inputs.attention_mask.to("cuda")).logits

    pred_ids = torch.argmax(logits, dim=-1)
    batch["pred_strings"] = processor.batch_decode(pred_ids)
    return batch

result = test_dataset.map(evaluate, batched=True, batch_size=8)

print("WER: {:2f}".format(100 * wer.compute(predictions=result["pred_strings"],
references=result["sentence"])))
```

Test Result: XX.XX % # TODO: write output of print here. IMPORTANT: Please remember to also replace {wer_result_on_test} at the top of with this value here. tags.

Training

The Common Voice `train`, `validation`, and ... datasets were used for training as well as ... and ... # TODO: adapt to state all the datasets that were used for training.

The script used for training can be found here # TODO: fill in a link to your training script here. If you trained your model in a colab, simply fill in the link here. If you trained the model locally, it would be great if you could upload the training script on github and paste the link here.

==========To here=============================>

Your model in then available under *huggingface.co/{your_username}/{your_chosen_xlsr-large_model_name}* for everybody to use 🥕.

How to evaluate my trained checkpoint

Having uploaded your model, you should now evaluate your model in a final step. This should be as simple as copying the evaluation code of your model card into a python script and running it. Make sure to note the final result on the model card **both** under the YAML tags at the very top **and** below your evaluation code under "Test Results".

Rules of training and evaluation

In this section, we will quickly go over what data is allowed to be used as training data, what kind of data preprocessing is allowed be used, and how the model should be evaluated.

To make it very simple regarding the first point: **All data except the official common voice `test` data set can be used as training data**. For models trained in a language that is not included in Common Voice, the author of the model is responsible to leave a reasonable amount of data for evaluation.

Second, the rules regarding the preprocessing are not that as straight-forward. It is allowed (and recommended) to normalize the data to only have lower-case characters. It is also allowed (and recommended) to remove typographical symbols and punctuation marks. A list of such symbols can *e.g.* be fonud here - however here we already must be careful. We should **not** remove a symbol that would change the meaning of the words, *e.g.* in English, we should not remove the single quotation mark `'` since it would change the meaning of the word `"it's"` to `"its"` which would then be incorrect. So the golden rule here is to not remove any characters that could change the meaning of a word into another word. This is not always obvious and should be given some

consideration. As another example, it is fine to remove the "Hypen-minus" sign " `-` " since it doesn't change the meaninng of a word to another one. *E.g.* " `fine-tuning` " would be changed to " `finetuning` " which has still the same meaning.

Since those choices are not always obvious when in doubt feel free to ask on Slack or even better post on the forum, as was done, *e.g.* here.

Tips and tricks

This section summarizes a couple of tips and tricks across various topics. It will continously be updated during the week.

How to combine multiple datasets into one

Check out this post.

How to effectively preprocess the data

How to do efficiently load datasets with limited ram and hard drive space

Check out this post.

How to do hyperparameter tuning

How to preprocess and evaluate character based languages

Further reading material

It is recommended that take some time to read up on how Wav2vec2 works in theory. Getting a better understanding of the theory and the inner mechanisms of the model often helps when fine-tuning the model.

However, if you don't like reading blog posts/papers, don't worry - it is by no means necessary to go through the theory to fine-tune Wav2Vec2 on your language of choice.

If you are interested in learning more about the model though, here are a couple of resources that are important to better understand Wav2Vec2:

- Facebook's Wav2Vec2 blog post
- Official Wav2Vec2 paper
- Official XLSR Wav2vec2 paper
- Hugging Face Blog
- How does CTC (Connectionist Temporal Classification) work

It helps to have a good understanding of the following points:

- How was XLSR-Wav2Vec2 pretrained? -> Feature vectors were masked and had to be predicted by the model; very similar in spirit to masked language model of BERT.
- What parts of XLSR-Wav2Vec2 are responsible for what? What is the feature extractor part used for? -> extract feature vectors from the 1D raw audio waveform; What is the transformer part doing? -> mapping feature vectors to contextualized feature vectors; ...
- What part of the model needs to be fine-tuned? -> The pretrained model **does not** include a language head to classify the contextualized features to letters. This is randomly initialized when loading the

pretrained checkpoint and has to be fine-tuned. Also, note that the authors recommend to **not** further fine-tune the feature extractor.

- What data was used to XLSR-Wav2Vec2? The checkpoint we will use for further fine-tuning was pretrained on **53** languages.
- What languages are considered to be similar by XLSR-Wav2Vec2? In the official XLSR Wav2Vec2 paper, the authors show nicely which languages share a common contextualized latent space. It might be useful for you to extend your training data with data of other languages that are considered to be very similar by the model (or you).

FAQ

- Can a participant fine-tune models for more than one language? Yes! A participant can fine-tune models in as many languages she/he likes
- Can a participant use extra data (apart from the common voice data)? Yes! All data except the official common voice `test data` can be used for training. If a participant wants to train a model on a language that is not part of Common Voice (which is very much encouraged!), the participant should make sure that some test data is held out to make sure the model is not overfitting.
- Can we fine-tune for high-resource languages? Yes! While we do not really recommend people to fine-tune models in English since there are already so many fine-tuned speech recognition models in English. However, it is very much appreciated if participants want to fine-tune models in other "high-resource" languages, such as French, Spanish, or German. For such cases, one probably needs to train locally and apply might have to apply tricks such as lazy data loading (check the "Lazy data loading" section for more details).