Speech Recognition using Whisper

Stars 553

Whisper is an automatic speech recognition (ASR) system trained on 680,000 hours of multilingual and multitask supervised data collected from the web. We show that the use of such a large and diverse dataset leads to improved robustness to accents, background noise, and technical language. Moreover, it enables transcription in multiple languages, as well as translation from those languages into English. We are open-sourcing models and inference code to serve as a foundation for building useful applications and for further research on robust speech processing. In this example, we will transcribe an audio clip locally, containerize the script and then run the container on Bacalhau.

The advantage of using Bacalhau over managed Automatic Speech Recognition services is that you can run your own containers which can scale to do batch process petabytes of videos or audio for automatic speech recognition

TD:LR

Using OpenAI whisper with Bacalhau to process audio files

Prerequisite

To get started, you need to install:

- Bacalhau client, see more information here
- Whisper,
- pytorch
- pandas

Running whisper locally

```
%%bash
pip install git+https://github.com/openai/whisper.git
pip install torch==1.10.1
pip install pandas
sudo apt update && sudo apt install ffmpeg
```

Before we create and run the script we need a sample audio file to test the code for that we download a sample audio clip.

```
%%bash
wget https://github.com/js-ts/hello/raw/main/hello.mp3
```

Create the script

We will create a script that accepts parameters (input file path, output file path, temperature, etc.) and set the default parameters. Also:

- If the input file is in mp4 format, then the script converts it to wav format.
- Save the transcript in various formats,
- We load the large model
- Then pass it the required parameters. This model is not only limited to English and transcription, it supports other languages and also does translation, into the following languages:

Next, let's create a openai-whisper script:

```
%%writefile openai-whisper.py
import argparse
import os
import sys
import warnings
import whisper
from pathlib import Path
import subprocess
import torch
import shutil
import numpy as np
parser = argparse.ArgumentParser(description="OpenAI Whisper Automatic Speech
Recognition")
parser.add_argument("-l",dest="audiolanguage", type=str,help="Language spoken
in the audio, use Auto detection to let Whisper detect the language. Select
from the following languages['Auto detection', 'Afrikaans', 'Albanian',
'Amharic', 'Arabic', 'Armenian', 'Assamese', 'Azerbaijani', 'Bashkir',
'Basque', 'Belarusian', 'Bengali', 'Bosnian', 'Breton', 'Bulgarian',
'Burmese', 'Castilian', 'Catalan', 'Chinese', 'Croatian', 'Czech', 'Danish',
'Dutch', 'English', 'Estonian', 'Faroese', 'Finnish', 'Flemish', 'French',
'Galician', 'Georgian', 'German', 'Greek', 'Gujarati', 'Haitian', 'Haitian
Creole', 'Hausa', 'Hawaiian', 'Hebrew', 'Hindi', 'Hungarian', 'Icelandic',
'Indonesian', 'Italian', 'Japanese', 'Javanese', 'Kannada', 'Kazakh', 'Khmer',
'Korean', 'Lao', 'Latin', 'Latvian', 'Letzeburgesch', 'Lingala', 'Lithuanian',
'Luxembourgish', 'Macedonian', 'Malagasy', 'Malay', 'Malayalam', 'Maltese',
'Maori', 'Marathi', 'Moldavian', 'Moldovan', 'Mongolian', 'Myanmar', 'Nepali',
'Norwegian', 'Nynorsk', 'Occitan', 'Panjabi', 'Pashto', 'Persian', 'Polish',
'Portuguese', 'Punjabi', 'Pushto', 'Romanian', 'Russian', 'Sanskrit',
'Serbian', 'Shona', 'Sindhi', 'Sinhala', 'Sinhalese', 'Slovak', 'Slovenian',
'Somali', 'Spanish', 'Sundanese', 'Swahili', 'Swedish', 'Tagalog', 'Tajik',
'Tamil', 'Tatar', 'Telugu', 'Thai', 'Tibetan', 'Turkish', 'Turkmen',
'Ukrainian', 'Urdu', 'Uzbek', 'Valencian', 'Vietnamese', 'Welsh', 'Yiddish',
'Yoruba'] ",default="English")
parser.add_argument("-p",dest="inputpath", type=str,help="Path of the input
file",default="/hello.mp3")
parser.add_argument("-v",dest="typeverbose", type=str,help="Whether to print
out the progress and debug messages. ['Live transcription', 'Progress bar',
'None']",default="Live transcription")
parser.add_argument("-g",dest="outputtype", type=str,help="Type of file to
generate to record the transcription. ['All', '.txt', '.vtt',
'.srt']",default="All")
parser.add_argument("-s",dest="speechtask", type=str,help="Whether to perform
X->X speech recognition (`transcribe`) or X->English translation
(`translate`). ['transcribe', 'translate']",default="transcribe")
parser.add_argument("-n",dest="numSteps", type=int,help="Number of
Steps",default=50)
parser.add_argument("-t",dest="decodingtemperature",
type=int,help="Temperature to increase when falling back when the decoding
fails to meet either of the thresholds below.",default=0.15 )
parser.add_argument("-b",dest="beamsize", type=int,help="Number of
Images",default=5)
parser.add_argument("-o",dest="output", type=str,help="Output Folder where to
store the outputs",default="")

args=parser.parse_args()
device = torch.device('cuda:0')
print('Using device:', device, file=sys.stderr)

Model = 'large'
whisper_model =whisper.load_model(Model)
video_path_local = os.getcwd()+args.inputpath
file_name=os.path.basename(video_path_local)
output_file_path=args.output

if os.path.splitext(video_path_local)[1] == ".mp4":
    video_path_local_wav =os.path.splitext(file_name)[0]+".wav"
    result  = subprocess.run(["ffmpeg", "-i", str(video_path_local), "-vn", "-
acodec", "pcm_s16le", "-ar", "16000", "-ac", "1", str(video_path_local_wav)])

# add language parameters
# Language spoken in the audio, use Auto detection to let Whisper detect the
language.
#  ['Auto detection', 'Afrikaans', 'Albanian', 'Amharic', 'Arabic',
'Armenian', 'Assamese', 'Azerbaijani', 'Bashkir', 'Basque', 'Belarusian',
'Bengali', 'Bosnian', 'Breton', 'Bulgarian', 'Burmese', 'Castilian',
'Catalan', 'Chinese', 'Croatian', 'Czech', 'Danish', 'Dutch', 'English',
'Estonian', 'Faroese', 'Finnish', 'Flemish', 'French', 'Galician', 'Georgian',
'German', 'Greek', 'Gujarati', 'Haitian', 'Haitian Creole', 'Hausa',
'Hawaiian', 'Hebrew', 'Hindi', 'Hungarian', 'Icelandic', 'Indonesian',
'Italian', 'Japanese', 'Javanese', 'Kannada', 'Kazakh', 'Khmer', 'Korean',
'Lao', 'Latin', 'Latvian', 'Letzeburgesch', 'Lingala', 'Lithuanian',
'Luxembourgish', 'Macedonian', 'Malagasy', 'Malay', 'Malayalam', 'Maltese',
'Maori', 'Marathi', 'Moldavian', 'Moldovan', 'Mongolian', 'Myanmar', 'Nepali',
'Norwegian', 'Nynorsk', 'Occitan', 'Panjabi', 'Pashto', 'Persian', 'Polish',
'Portuguese', 'Punjabi', 'Pushto', 'Romanian', 'Russian', 'Sanskrit',
'Serbian', 'Shona', 'Sindhi', 'Sinhala', 'Sinhalese', 'Slovak', 'Slovenian',
'Somali', 'Spanish', 'Sundanese', 'Swahili', 'Swedish', 'Tagalog', 'Tajik',
'Tamil', 'Tatar', 'Telugu', 'Thai', 'Tibetan', 'Turkish', 'Turkmen',
'Ukrainian', 'Urdu', 'Uzbek', 'Valencian', 'Vietnamese', 'Welsh', 'Yiddish',
'Yoruba']
language = args.audiolanguage
# Whether to print out the progress and debug messages.
# ['Live transcription', 'Progress bar', 'None']
verbose = args.typeverbose
#  Type of file to generate to record the transcription.
# ['All', '.txt', '.vtt', '.srt']
```

```
output_type = args.outputtype
# Whether to perform X->X speech recognition (`transcribe`) or X->English
translation (`translate`).
# ['transcribe', 'translate']
task = args.speechtask
# Temperature to use for sampling.
temperature = args.decodingtemperature
#  Temperature to increase when falling back when the decoding fails to meet
either of the thresholds below.
temperature_increment_on_fallback = 0.2
#  Number of candidates when sampling with non-zero temperature.
best_of = 5
#  Number of beams in beam search, only applicable when temperature is zero.
beam_size = args.beamsize
# Optional patience value to use in beam decoding, as in [*Beam Decoding with
Controlled Patience*](https://arxiv.org/abs/2204.05424), the default (1.0) is
equivalent to conventional beam search.
patience = 1.0
# Optional token length penalty coefficient (alpha) as in [*Google's Neural
Machine Translation System*](https://arxiv.org/abs/1609.08144), set to
negative value to uses simple length normalization.
length_penalty = -0.05
# Comma-separated list of token ids to suppress during sampling; '-1' will
suppress most special characters except common punctuations.
suppress_tokens = "-1"
# Optional text to provide as a prompt for the first window.
initial_prompt = ""
# if True, provide the previous output of the model as a prompt for the next
window; disabling may make the text inconsistent across windows, but the model
becomes less prone to getting stuck in a failure loop.
condition_on_previous_text = True
#  whether to perform inference in fp16.
fp16 = True
#  If the gzip compression ratio is higher than this value, treat the decoding
as failed.
compression_ratio_threshold = 2.4
# If the average log probability is lower than this value, treat the decoding
as failed.
logprob_threshold = -1.0
# If the probability of the <|nospeech|> token is higher than this value AND
the decoding has failed due to `logprob_threshold`, consider the segment as
silence.
no_speech_threshold = 0.6

verbose_lut = {
    'Live transcription': True,
    'Progress bar': False,
    'None': None
}

args = dict(
    language = (None if language == "Auto detection" else language),
    verbose = verbose_lut[verbose],
    task = task,
    temperature = temperature,
    temperature_increment_on_fallback = temperature_increment_on_fallback,
    best_of = best_of,
    beam_size = beam_size,
    patience=patience,
    length_penalty=(length_penalty if length_penalty>=0.0 else None),
    suppress_tokens=suppress_tokens,
    initial_prompt=(None if not initial_prompt else initial_prompt),
    condition_on_previous_text=condition_on_previous_text,
    fp16=fp16,
    compression_ratio_threshold=compression_ratio_threshold,
    logprob_threshold=logprob_threshold,
    no_speech_threshold=no_speech_threshold
)

temperature = args.pop("temperature")
temperature_increment_on_fallback =
args.pop("temperature_increment_on_fallback")
if temperature_increment_on_fallback is not None:
    temperature = tuple(np.arange(temperature, 1.0 + 1e-6,
temperature_increment_on_fallback))
else:
    temperature = [temperature]

if Model.endswith(".en") and args["language"] not in {"en", "English"}:
    warnings.warn(f"{Model} is an English-only model but receipted
'{args['language']}'; using English instead.")
    args["language"] = "en"

video_transcription = whisper.transcribe(
    whisper_model,
    str(video_path_local),
    temperature=temperature,
    **args,
)

# Save output
writing_lut = {
    '.txt': whisper.utils.write_txt,
    '.vtt': whisper.utils.write_vtt,
    '.srt': whisper.utils.write_txt,
}

if output_type == "All":
    for suffix, write_suffix in writing_lut.items():
        transcript_local_path
=os.getcwd()+output_file_path+'/'+os.path.splitext(file_name)[0] +suffix
        with open(transcript_local_path, "w", encoding="utf-8") as f:
            write_suffix(video_transcription["segments"], file=f)
        try:
            transcript_drive_path =file_name
        except:
            print(f"**Transcript file created: {transcript_local_path}**")
else:
    transcript_local_path =output_file_path+'/'+os.path.splitext(file_name)[0]
+output_type

    with open(transcript_local_path, "w", encoding="utf-8") as f:
        writing_lut[output_type](video_transcription["segments"], file=f)
```

Let's run the script with the default parameters:

```
%%bash
python openai-whisper.py
```

Viewing the outputs

```
%%bash
cat hello.srt
```

Containerize Script using Docker

To build your own docker container, create a `Dockerfile`, which contains instructions on how the image will be built, and what extra requirements will be included.

```
FROM  pytorch/pytorch:1.12.1-cuda11.3-cudnn8-runtime

WORKDIR /

RUN apt-get -y update

RUN apt-get -y install git

RUN python3 -m pip install --upgrade pip

RUN python -m pip install regex tqdm Pillow

RUN pip install git+https://github.com/openai/whisper.git

ADD hello.mp3 hello.mp3

ADD openai-whisper.py openai-whisper.py

RUN python openai-whisper.py
```

We choose `pytorch/pytorch:1.12.1-cuda11.3-cudnn8-runtime` as our base image

And then install all the dependencies, after that we will add the test audio file and our openai-whisper script to the container, we will also run a test command to check whether our script works inside the container and if the container builds successfully

INFO

See more information on how to containerize your script/app here

Build the container

We will run `docker build` command to build the container;

```
docker build -t <hub-user>/<repo-name>:<tag> .
```

Before running the command replace;

- **hub-user** with your docker hub username, If you don’t have a docker hub account follow these instructions to create a Docker account, and use the username of the account you created
- **repo-name** with the name of the container, you can name it anything you want
- **tag** this is not required but you can use the latest tag

In our case

```
docker build -t jsacex/whisper
```

Push the container

Next, upload the image to the registry. This can be done by using the Docker hub username, repo name or tag.

```
docker push <hub-user>/<repo-name>:<tag>
```

In our case

```
docker push jsacex/whisper
```

Running a Bacalhau Job

We will transcribe the moon landing video, which can be found here: https://www.nasa.gov/multimedia/hd/apollo11_hdpage.html

Since the downloaded video is in mov format we convert the video to mp4 format and then upload it to our public storage in this case IPFS. We will be using NFT.Storage (Recommended Option). To upload your dataset using NFTup just drag and drop your directory it will upload it to IPFS

After the dataset has been uploaded, copy the CID:

`bafybeielf6z4cd2nuey5arckect5bjmelhouvn5rhbjlvpvhp7erkrc4nu`

To submit a job, run the following Bacalhau command:

```
%%bash --out job_id
bacalhau docker run \
    --id-only \
    --gpu 1 \
    --timeout 3600 \
    --wait-timeout-secs 3600 \
    jsacex/whisper \
    -i ipfs://bafybeielf6z4cd2nuey5arckect5bjmelhouvn5rhbjlvpvhp7erkrc4nu \
    -- python openai-whisper.py -p inputs/Apollo_11_moonwalk_montage_720p.mp4
-o outputs
```

```
%env JOB_ID={job_id}
```

Structure of the command

Let's look closely at the command above:

- `-i ipfs://bafybeielf6z4cd2nuey5arckect5bjmelhouvn5r`: flag to mount the CID which contains our file to the container at the path `/inputs`
- `-p inputs/Apollo_11_moonwalk_montage_720p.mp4` : the input path of our file
- `-o outputs`: the path where to store the outputs
- `--gpu` : here we request 1 GPU
- `jsacex/whisper`: the name and the tag of the docker image we are using

Checking the State of your Jobs

- **Job status**: You can check the status of the job using `bacalhau list`.

```
%%bash
bacalhau list --id-filter ${JOB_ID} --wide
```

When it says `Published` or `Completed`, that means the job is done, and we can get the results.

- **Job information**: You can find out more information about your job by using `bacalhau describe`.

```
%%bash
bacalhau describe ${JOB_ID}
```

- **Job download**: You can download your job results directly by using `bacalhau get`. Alternatively, you can choose to create a directory to store your results. In the command below, we created a directory and downloaded our job output to be stored in that directory.

```
%%bash
rm -rf results && mkdir -p results
bacalhau get $JOB_ID --output-dir results
```

Viewing your Job Output

To view the file, run the following command:

```
%%bash
cat results/outputs/Apollo_11_moonwalk_montage_720p.vtt
```

Edit this page

*Last updated on **Jan 11, 2024***