Tab 1

Meskvlla33's

Guide to Training Audio Source Separation Models

(aka how to train a vocal remover)

0 - Overview

Why would you want to train / fine-tuning a model?

It can help in such instances that a model does not satisfy your requirements for separating whatever it is you listen to. In the case of the doc writer, metal, most separation methods proved to be very hit or miss with this type of music because there's so many overlapping sounds that it makes it difficult for that model to correctly separate the music. Roformers do a pretty decent job, sure, but they also come with tradeoffs.

What do you need for training?

A computer with an NVIDIA GPU (required), at least 8gb of VRAM or more (more is better) and a somewhat fast card. There are ways to optimize training further but this is the basics of what you need
Or alternatively you can hire fast GPUs with cloud solutions such as vast.ai, runpod.io etc. It all comes down to if you have a powerful card, if it's like 8gb, could be fine for pretraining or shifting a model's target, but actual training forget it, it's better to go cloud.

You also will probably need python to train locally, grab it from here:
https://www.python.org/downloads/

Can I train without any prior knowledge?

Yes, of course. This is the job of this document, because, at least on deton's document the training sections is useful but it does not provide a real "how to". I know it could've helped me a lot.

Does this document include "everything"?

Well, yes and no. It does include at least everything I have learned over the course of several months and talking with users from the server, but there's still things I have yet to learn probably!

With this out of the way, let's get into it shall we!

1 - Requirements:

These are the things you'll need for training a model:

- A dataset ***(composed of official "whatever-it-is-that-you-want-to-train", in my case it is Instrumentals and Vocals, so my dataset has official instrumentals and vocals inside)***
- A validation dataset:
    - Stem 1
    - Stem 2
    - Mixture ***(Stem 1 + 2 joined together)***
- The training repository
- Models you want to train. The training repo already has some, but you can train any models you want as long as it has a checkpoint and a yaml ***(we'll come back to these later, if you use UVR you already know what these are)***

2 - How to: Dataset

If you do not have a dataset at your disposition already, I will just leave this video by Bas Curtiz. It explains EVERYTHING you need to know about creating a dataset, so I will not be detailing all the steps here in my document:
https://www.youtube.com/watch?v=Wmt_0zu94L8

Note: make sure the tracks inside the datasets are either in .flac or .wav. MP3 will not work

The video shows only one pair of a dataset, but really there are two types of dataset you'd want to know:

Type 1 dataset:

> [name of the folder, "MAIN_DATASET" or something]

  > [folder: song name 1]

    > bass.flac

    > drums.flac

    > other.flac

    > vocals.flac

  > [folder: song name 2]

    > bass.flac

    > drums.flac

    > other.flac

    > vocals.flac

The Type 1 dataset is based on the MUSDB18 dataset and is used for 4 stems models.

Type 2 dataset:

> [name of the folder, "MAIN_DATASET" or something]

> [folder: other]

> song 1 (Instrumental).flac
> song 2 (Instrumental).flac
> song 3 (Instrumental).flac
> song 4 (Instrumental.flac

> [folder: vocals]

> song 1 (Vocals).flac
> song 2 (Vocals).flac
> song 3 (Vocals).flac
> song 4 (Vocals).flac

The Type 2 dataset has only 2 folders (here "other" and "vocals"), and you dump all the instrumentals you have into "other", same for "vocals".

The song length for type 2 dataset doesn't need to match because they are selected randomly, split in chunks (just like when doing a separation in UVR), mixed together and entered into the model.

deton's doc recommends at least "200 audio files" for training, but more is better. For example, if you take the final version of the Metal Dataset that I am currently training on, it has ***2135 instrumentals and 1779 vocals*** (total 3914 tracks).

In some cases, like if your training data is 10 minutes + (for each file), you can get away with a lower file count, because the model will still chunk the audio (in the audio => chunk_size section of the yaml).

They need to be from official sources, otherwise for some things like vocals you could use Mel-Roformer or inverting to get proper vocals for that song, but I would recommend having official instrumentals + vocals or whatever it is you want to train.

Both datasets accept any audio length. For type 1 tho, make sure they end at the same time, otherwise it will throw you an error when fetching the metadata (***"Warning: lengths of stems are different for path: [C:\PATH_TO_DATASET\SONG_FOLDER]. (25666810 != 28057480"***)

I've also encountered this error while training SCNet (from 4 stems to 2 stems), make sure the target_instrument is set to *null* and instruments display

- *vocals*
- *other*

(***"RuntimeError: output with shape [1, 2, {chunk size}] doesn't match with the broadcast shape [2, 2, {chunk size}]"***)

The dataset will probably take the longest time to make, because you need to scour the internet for official stems or instrumentals

3 - How to: Validation Dataset

For the validation dataset, it needs to follow this general layout:

> [name of the folder, “validation_dataset” or something]
  > [folder: song name 1]
    > mixture.wav
    > other.wav
    > vocals.wav
  > [folder: song name 2]
    > mixture.wav
    > other.wav
    > vocals.wav

This is for Type 2 datasets, for Type 1 it would also include *drums.wav* + *bass.wav*

The validation dataset only accepts 16 bit wav files, so make sure to export them correctly, i’ve had errors thrown because it was in flac.

Use tracks that are already inside your dataset because they will be easier to grab once you need to do the validation dataset.

Validation metrics will be higher when you use full tracks. But you can also make the validation faster by doing 30 seconds to ~1 minute audio clips if it takes too long (be aware: doing this will decrease metrics)

Make sure all of the audio ends at the same time. Otherwise it will throw this error at you during the validation stage:
ValueError: operands could not be broadcast together with shapes

4 - How to: The Training Repository

This is the link to the training repo:
https://github.com/ZFTurbo/Music-Source-Separation-Training
(to download it on your computer: press the green “Code” button and then click on “Download as ZIP”)

Extract the repo to whatever location.

The first thing you’d wanna do is download one model of your choice that you want to train, for example:

Let's say you want to finetune Kimberley's Roformer, you'd go onto the repo, scroll down to the "Pre-trained models" section, click on "List of Pre-trained models", grab the "Config" and the "Weights" from both the links at your left.

(on some occasions, "Configs" might not download but open in your browser, open notepad++, enter the code and then save it as .yaml)

Config = **yaml**, where every information about your model is stored, we will come back to it
Weights = **checkpoint**, the model itself
All of the models included on the repo are open-source so be weary of that

Next, you need to create a folder called "***results***" and place the checkpoint inside.
Place the yaml where *train.py / valid.py / inference.py* is located (root of the repo)

Note: Roformers have their own spectrogram loss when trained

5 - A section about commands and arguments

Since this is a python script, it will of course need commands to run
These are most of the useful commands (you'd probably come back to this section a lot)

COMMANDS:

IMPORTANT NOTICE: RUN THESE COMMANDS WITH ADMINISTRATOR ACCESS TO THE COMMAND PROMPT / POWERSHELL FOR LOCAL USE

Installing pytorch:
https://pytorch.org/get-started/locally/ <= (grab the command on here)

Installing the requirements:
pip install -r requirements.txt

Installing other requirements: *(if it didn't install when running the previous command):*
pip install [name-of-requirement]

Changing directory: *(if you're in another folder for example)*
cd [path-to-folder]

Starting training:
python train.py --model_type [TYPE OF MODEL] --config_path [YAML] --results_path results/ --data_path [PATH TO DATASET] --dataset_type [1, 2, 3, 4] --num_workers 4 --device_ids 0 --start_check_point results/[CHECKPOINT] --valid_path [PATH TO VALIDATION DATASET] --metric_for_scheduler sdr --metrics [OTHER METRICS YOU NEED]

Note: you WILL have to change everything in purple according to your needs, for the metrics side of things, you'll need to look at the arguments part of this section. I would recommend copying this and using a text editor to hold onto it.

Alternatively, you could edit it using the base Windows notepad, copy it and editing the arguments there and then save it as "run.bat" on the root of the repo, that way you can run it in 2 clicks without the need for copying it onto the command prompt every time. Of course, you can still use it with the command prompt by running ".\run.bat".

Inferencing a model: *(doing a separation with a model)*
python inference.py --model_type [TYPE OF MODEL] --config_path [YAML] --start_check_point results/[CHECKPOINT]--input_folder input --store_dir separation_results

Note: you will have to create 2 folders: "input", where you will drop your desired file you wish to separate and "separation_results", where the script will put the separated audio

(FOR CLOUD ONLY) **Troubleshooting the OpenBLAS error when attempting to train:**
export OPENBLAS_NUM_THREADS=1

Note: this will only do it for your current session, but we'll dive into that later if you're going cloud

ARGUMENTS:

There are several arguments you can include for your training and inference commands.

Training:

--model_type: ***type of models, usually: "mdx23c | htdemucs | segm_models | mel_band_roformer | bs_roformer | swin_upernet | bandit"***
--config_path: ***path to config file***
--start_check_point: ***initial checkpoint to start training***
--results_path: ***path to the folder where results will be stored (both .ckpt files and the metadata)***
--data_path: ***path to your dataset folder***
--dataset_type: ***dataset type. must be 1, 2, 3 or 4, details here:***
https://github.com/ZFTurbo/Music-Source-Separation-Training/blob/main/docs/dataset_types.md
--valid_path: ***path to your validation dataset folder***
--num_workers: ***controls how many CPU cores are used to load and process the data***
--pin_memory : ***controls the number of parallel data-loading workers***
--seed: ***controls the randomness in the training process, experiment with numbers***
--device_ids: ***list of gpu IDs, usually 0***
--use_multistft_loss: ***use MultiSTFT (Short-Time Fourier Transform) loss, spectrogram based***

--use_mse_loss: ***use default MSE loss, waveform based***
--use_l1_loss: ***use L1 loss***
--wandb_key: ***WandB (Weights and Biases) API Key***
--pre_valid: ***runs validation before training***
--metrics: ***types of metrics you can use: [sdr | l1_freq | si_sdr | neg_log_wmse | aura_stft | aura_mrstft | bleedless | fullness]***
--metric_for_scheduler: ***types of metrics that the scheduler will use to change the learning rate of the model: [sdr | l1_freq | si_sdr | neg_log_wmse | aura_stft | aura_mrstft | bleedless | fullness]***
--train_lora: ***Train with LoRA (Low Rank Adaptation)***
--lora_checkpoint: ***Initial checkpoint for LoRA weights***

Metrics:

sdr : Signal to Distortion Ratio, yk what this is lol
l1_freq: L1 Frequency, similar to sdr, spectrogram based
si_sdr: Scale Invariant Signal to Distortion Ratio, basically sdr but it ignores the scaling between the target and noise
neg_log_wmse: Negative Log Weighted Mean Square Error, another loss function
aura_stft: Aura Short-Time Fourier Transform, focuses on the perceptual quality of separated audio
aura_mrstft: Aura Multi Resolution Short-Time Fourier Transform, a more advanced aura_stft on multi resolutions
bleedless: tells you how much bleed of one output is in the other (ex: tells how much vocal bleed there is in the instrumental), spectrogram based
fullness: tells you how full the target instrument is, spectrogram based

Remember, metrics are NUMBERS and should be only for a quick evaluation. Listen to the outputs produced by the model and see if it sounds good to you.

Inference:

--model_type: ***type of models, usually: "mdx23c | htdemucs | segm_models | mel_band_roformer | bs_roformer | swin_upernet | bandit"***
--config_path: ***path to config file***
--start_check_point: ***path to a checkpoint***
--input_folder: ***folder with mixtures to process***
--store_dir: ***path to store results as wav file***
--draw_spectro: ***this code will generate spectrograms for the resulting stems. the value defines for how many seconds os track spectrogram will be generated (default is 0)***
--device_ids: ***list of gpu ids***
--extract_instrumental: ***invert vocals to get instrumental if provided (will output vocals if it is an instrumental model)***
--disable_detailed_pbar: ***disables the progress bar***
--force_cpu: ***forces the use of the CPU even when CUDA is available (won't use your GPU)***
--flac_file: ***outputs FLAC files instead of wav***

--pcm_type: **PCM type for FLAC files (PCM_16 or PCM_24)**
--use_tta: ***Flag that adds test time augmentation during inference (polarity and channel inversion). While this triples the runtime, it reduces noise and slightly improves prediction quality.***
--lora_checkpoint: ***Initial checkpoint for LoRA weights***

6 - YAMLs and optimal parameters

The configs of the models can be modified to whatever you need

A yaml has 4 sections:

- audio
- model
- training
- inference

I will focus on the training parameters here, audio and models you don't need to change too much, except maybe the audio batch size for conversions (both for the training repo and UVR; list here):

dim_t to *chunk_size* conversion for roformer models:
256 => 112455 (2.55 seconds)
801 => 352800 (8.00 seconds)
1101 => 485100 (11.00 seconds)
1333 => 587412 (13,22 seconds)

[formula is ***chunk_size = (dim_t -1) x hop_length***] (thanks jarredou!!!)

training:
  batch_size: 1 ***(number of audio samples used for updating the models weights, higher batch sizes use more vram)***
  gradient_accumulation_steps: 1 ***(simulates a larger batch size without the vram requirement, batch_size becomes "batch_size x gradient_accumulation_steps", models weights won't be updated for gradient_accumulation_steps)***
  grad_clip: 0
  instruments: ***(your specific instruments inside the dataset, change them accordingly)***
  - vocals
  - other
  lr: 1.0e-05 ***(learning rate, another common example is 5.0e-06)***
  patience: 2 ***(how long the model will stay on that learning rate before it is reduced, set to 2 epochs here)***
  reduce_factor: 0.95 ***(how much the learning rate will be reduced)***
  target_instrument: vocals ***(targets the desired stem that the model will focus on)***

num_epochs: 1000 ***(number of times you will train for num_steps before training stops, can be changed when loading from checkpoints)***
num_steps: 1000 ***(number of times the models weights will be updated per epoch (divided by gradient_accumulation_steps), validation will be done every time num_steps is reached, can be changed when loading from checkpoints)***
  augmentation: false ***(enables augmentations by audiomentations and pedalboard)***
  augmentation_type: null
  use_mp3_compress: false ***(deprecated)***
  augmentation_mix: false ***(mix several stems of the same type with some probability)***
  augmentation_loudness: false ***(randomly change loudness of each stem)***
  augmentation_loudness_type: 1 ***(type 1 or 2)***
  augmentation_loudness_min: 0
  augmentation_loudness_max: 0
  q: 0.95
  coarse_loss_clip: false
  ema_momentum: 0.999
  optimizer: adam ***(optimizer for training)***
  other_fix: true ***(it's needed for checking on multisong dataset if other is actually instrumental)***
  use_amp: true ***(enable or disable usage of mixed precision (float16) - usually it must be true)***
  use_torch_checkpoint: true ***(uses gradient checkpointing, saves vram when training on lower spec machines)***

Inference ***num_overlap*** should always be set to 1 when training (to save time).

Optimal parameters for yaml takes time to figure out, so feel free to experiment with them.
Include ***use_torch_checkpoint*** if you truly need it, or if it outputs CUDA_OutOfMemory errors

7 - How To: Fullness Models + Model Fusion script (by Sucial)

To train models for fullness, you'll need 3 steps:

1 - in train.py, there is a line titled this: "***if args.model_type in ['mel_band_roformer', 'bs_roformer']:***', place a 1 before either mel or bs [or search anything with "roformer" inside]

2 - use '***--use_multistft_loss***' as your loss argument in the training command

3 - add this to your config (you can play with these if youd like):

```
loss_multistft:
  fft_sizes:
  - 1024
  - 2048
  - 4096
  hop_sizes:
  - 512
  - 1024
  - 2048
  win_lengths:
  - 1024
  - 2048
  - 4096
  window: "hann_window"
  scale: "mel"
  n_bins: 128
  sample_rate: 44100
  perceptual_weighting: false
  w_sc: 0.0
  w_log_mag: 1.0
  w_lin_mag: 0.0
  w_phs: 0.0
  mag_distance: "L1"
```

WARNING: Training like this will give fullness models but with lots of noise!

For the Model Fusion script, you can find it here:
https://huggingface.co/Sucial/Dereverb-Echo_Mel_Band_Roformer/blob/main/scripts/model_fusion.py
(this is not exactly training related, but it still can be useful)

To use it, you'd have to download the .py file and place it into MSST.

Requirements:
- All checkpoints must have the same target, you CANNOT fuse a vocal model with an instrumental model, for example: all the stems would be "other" if you want to fuse 2 instrumental models
- (NOT CONFIRMED) Possibly both checkpoints must have the same 'base', example: if you trained a model with dim 384, depth 6 and mask_estimator_depth 2, both ckpt should match

Replace the 'model_1.ckpt' in the script with the actual yaml name of the checkpoint, same for the others
This is weight based and all of them should equal to 1, for example i have them as 0.5 and 0.5 because I'm still testing around with these.

8 - Training from Scratch

I wanted to add this before going into training specifically, because it is important to know.

You CAN train from no checkpoints if for some reason fine-tuning a model doesn't suit your needs and you need a model for your SPECIFIC case.
(example: I am currently pre-training a model on my own metal dataset using my laptop then I will use the cloud to let it go a bit faster and with different parameters, so that in the summer I have some sort of a "decent" base model. Other models have always been hit or miss for me)
[THIS EXAMPLE IS NOT WHAT I'M DOING ANYMORE BUT IM STILL LEAVING IT]

To do this, just remove the --start_check_point argument from your training command.

BE WARNED: The metrics WILL be bad, but only because you're starting from nothing. If you use a laptop like me they probably will be even worse because of the limitation of the hardware If you use cloud, it will go more smoothly, just remember to use a smaller chunk_size until it converges (aka gets better at separating) then you could use another chunk size

Example: my roformer with batch size 4 takes up the entire vram of an H200 (which is 140 gb)

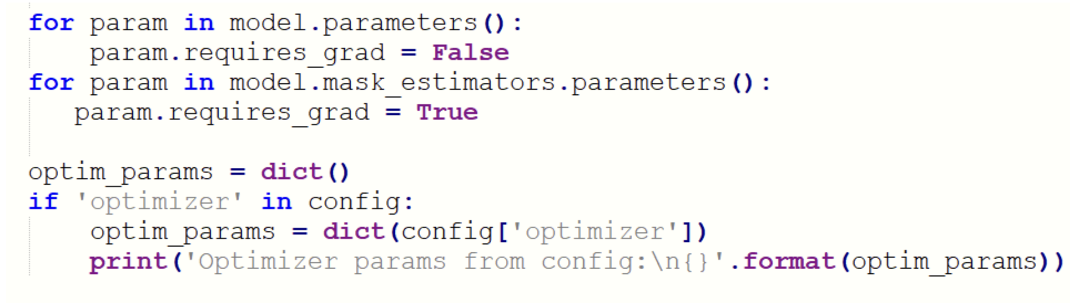
9 - Special Cases: How to shift focus from one stem to another

I've had a case where I needed to shift focus from vocals to instrumentals and after many unsuccessful tries, unwa jumped in to save the day.

He proposed that i'd try a "transfer learning" method, and he gave me this piece of code:

```
for param in model.parameters():
     param.requires_grad = False
for param in model.mask_estimators.parameters():
    param.requires_grad = True
```

It needs to look like this:

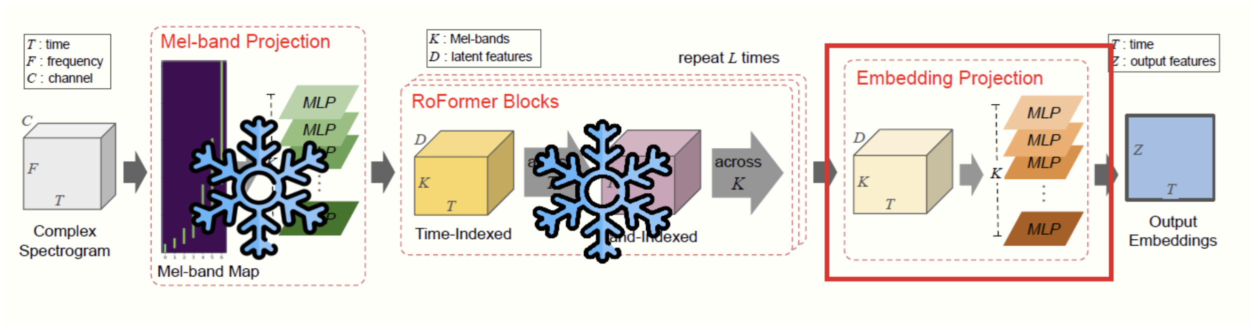


```
for param in model.parameters():
    param.requires_grad = False
for param in model.mask_estimators.parameters():
   param.requires_grad = True

optim_params = dict()
if 'optimizer' in config:
    optim_params = dict(config['optimizer'])
    print('Optimizer params from config:\n{}'.format(optim_params))
```

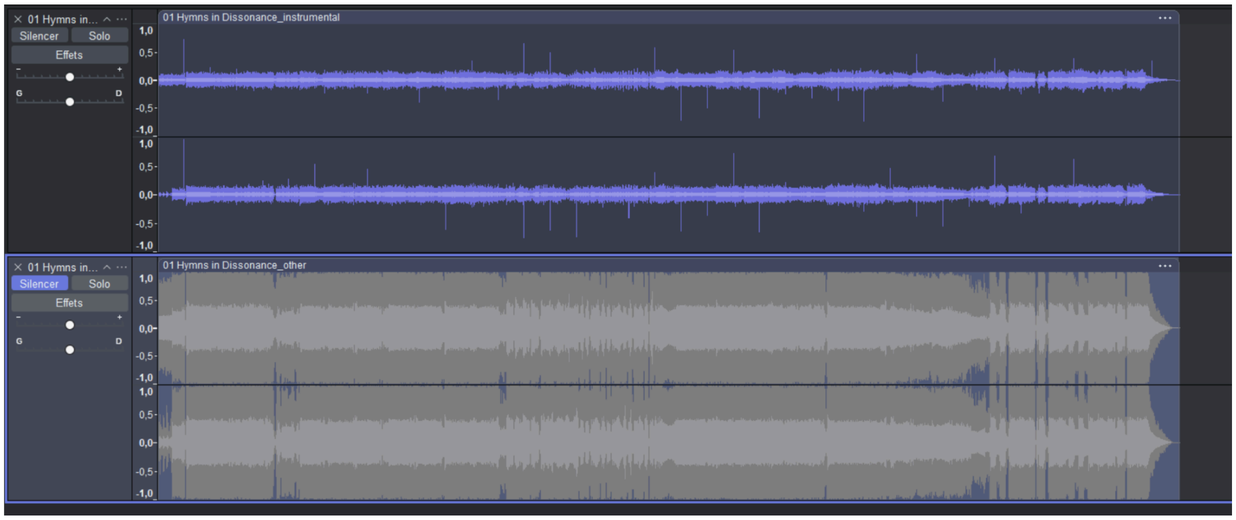
You'll need to put it above line 203 in ***train.py***: *optim_params = dict()*

unwa said that “in my experience, it is the MaskEstimator that changes significantly when the target is changed”.
What this code does is it freezes the Mel-band Projections and the Roformer Blocks of the model (Mel-Roformer) and just changes the last modules of the training to switch from vocals to instrumental, as explained by this graph:

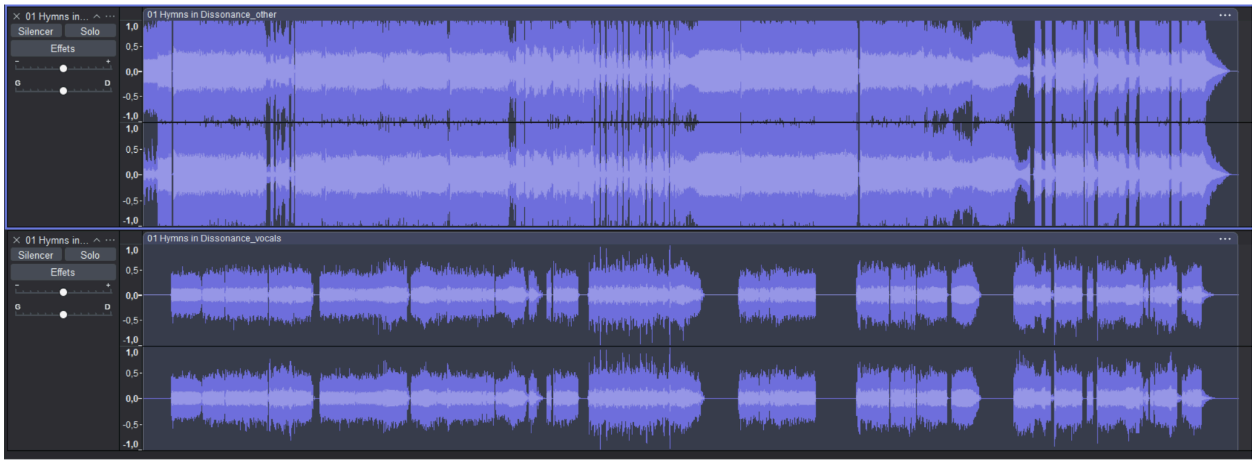


Note that while this significantly reduces the training time, it will not finetune the other blocks, and it doesn’t always work in all cases. In such cases, remove the code off train.py. It is fine for switching but not for full training.
I pretrained my model until epoch 12 to ensure a full switch, and a further 2 epochs to ensure it would stay onto the instrumental (because I was skeptical).

Be warned that if you put your learning rate too fast when further training, it can do weird stuff and say example: “inst other fullness: 49” etc, that’s because it learned too fast (I think). Like this for example:



It’s supposed to look like this:

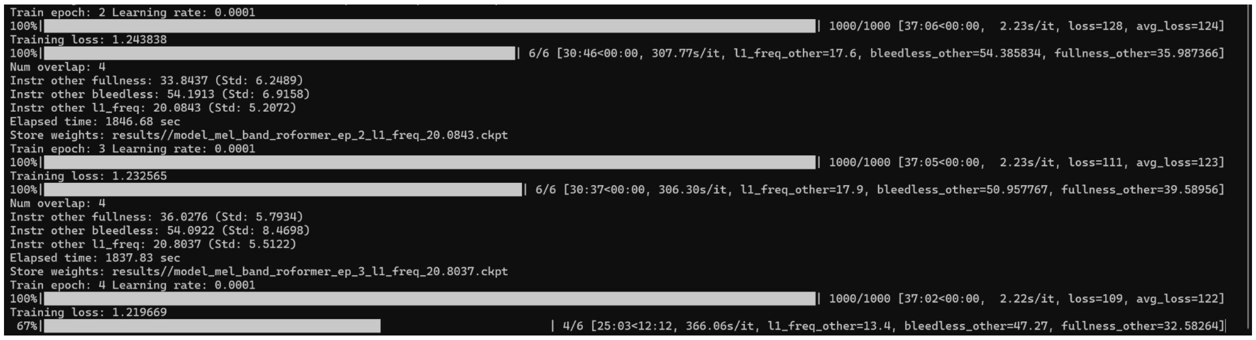


10 - How to: Train

a) Local Training

Training locally is simple, you just open the command prompt, pop in the training command and let it do the rest.
Your gpu will spin real fast, that’s how you know it works lol

Example of a training screenshot:



```
Train epoch: 2 Learning rate: 0.0001
100%|████████████████████████████████████████████████| 1000/1000 [37:06<00:00,  2.23s/it, loss=128, avg_loss=124]
Training loss: 1.243838
100%|██████████████████████████████| 6/6 [30:46<00:00, 307.77s/it, l1_freq_other=17.6, bleedless_other=54.385834, fullness_other=35.987366]
Num overlap: 4
Instr other fullness: 33.8437 (Std: 6.2489)
Instr other bleedless: 54.1913 (Std: 6.9158)
Instr other l1_freq: 20.0843 (Std: 5.2072)
Elapsed time: 1846.68 sec
Store weights: results//model_mel_band_roformer_ep_2_l1_freq_20.0843.ckpt
Train epoch: 3 Learning rate: 0.0001
100%|████████████████████████████████████████████████| 1000/1000 [37:05<00:00,  2.23s/it, loss=111, avg_loss=123]
Training loss: 1.232565
100%|██████████████████████████████| 6/6 [30:37<00:00, 306.30s/it, l1_freq_other=17.9, bleedless_other=50.957767, fullness_other=39.58956]
Num overlap: 4
Instr other fullness: 36.0276 (Std: 5.7934)
Instr other bleedless: 54.0922 (Std: 8.4698)
Instr other l1_freq: 20.8037 (Std: 5.5122)
Elapsed time: 1837.83 sec
Store weights: results//model_mel_band_roformer_ep_3_l1_freq_20.8037.ckpt
Train epoch: 4 Learning rate: 0.0001
100%|████████████████████████████████████████████████| 1000/1000 [37:02<00:00,  2.22s/it, loss=109, avg_loss=122]
Training loss: 1.219669
 67%|████████████████████          | 4/6 [25:03<12:12, 366.06s/it, l1_freq_other=13.4, bleedless_other=47.27, fullness_other=32.58264]
```

Train epoch lists the epoch it is training right now, it starts from 0 and then goes for as long as you want
Learning rate is your learning rate, here set to 1.0e-04

First progress bar is your 1000 step training bar, then it lists the “Time passed < Estimated Time” for the epoch to complete.
2.23 here is seconds per iteration (how fast it goes).
loss measures how different the output is to the original, you want that to be as low as possible
avg_loss is just the average training loss

Second progress bar is your validation data progress bar, here it's very long because I used full tracks for my validation (and i only had 6)
6/6 is your number of tracks inside the validation number
"Time passed < Estimated Time" for the validation
Seconds per iteration for validation
l1_freq_other lists the l1 frequency of the track it is validating, here track 4 (for epoch 3), it's at 13.4
bleedless_other and lists the bleedless of that track's instrumental, fullness_other is the same

Num(ber) overlap is your INFERENCE overlap number, here set to 4 (set it to 1, it'll be much quicker)

Inst_other_fullness / bleedless / l1_freq / sdr lists the average results of the instrumental stem if you were to convert a track using this checkpoint.

One thing to know is that for some reason:
- If fullness increases, bleedless decreases
- If bleedless increases, fullness decreases

so you might want to take this into consideration

What else to do?
Well, training is a waiting game, and if you're training locally you cannot really spin up a game to play because you will use your GPU, so you might want to do something else because training is very boring.

However, that might not be the case if you're going for the other option:

b) - Cloud training

This is a step by step tutorial on how to use vast.ai, because it is really not simple to use and while there's information out there, it's not really representative of our use case here.

Be warned, these instructions might be a bit confusing!

1 - Put the training repo + both datasets on Google Drive inside a folder of your preference (and make sure to have at least 100gb of storage).
Vast.ai offers a direct connection to your drive in the 'CLOUD' section on the main website (below INSTANCES), click on the 'Connect Google Drive' button, name the connection whatever name and make the connection.

2 - Select a GPU to rent in the 'SEARCH' section, make sure to take the 'PyTorch (cuDNN Runtime)' template, with Cuda 12.4, SSH and Jupyter. Make sure to have enough disk space as well, but that will also change the total price of your rent.

3 - Once the instance is created, it will pop up in the 'INSTANCES' section. Wait for it to load. Enter it, and create a folder where it can dump all of the data from your drive in. ***(mine is just called TRAINING_STUFF)***. Do not close this page, return to 'INSTANCES' and click on the cloud icon on the instance. It will show a message saying 'Upload/Download Data from Cloud Providers'. Enter your folder path (***example: if you put the training data in a folder called test, the path would be '/test')***.

Do the same for the output path, in my case it's /TRAINING_STUFF. Let it do its thing.

4 - When it's done, you should have all of your data inside your Jupyter Notebook. Copy the training command from earlier in this document, and add the data path to the dataset and validation dataset AND MAKE SURE TO ADD A SLASH before and after the path. Example: '--data_path /TRAINING_STUFF/DATA/2stem_metal_dataset/' would be the full argument for the dataset path. Once it's all compete, click on '*File'* > *'New'* > *'Terminal'*

(this next part will be really annoying!) Paste in these commands in the terminal in sequence (one after the other):

- **Change directory**
- **Install requirements**
- **OpenBLAS troubleshooting command** *[this appeared when I was at the metadata stage, but it turned out to be something else entirely separate from this error and idk what it is]*
- **Training command**

if it shows an error (*no_module_named[x]*), do ***pip install [x]*** + repaste training command every time until you see the 'Collecting Metadata' part. Once it's done, it should start collecting the metadata and start the training.

all of the missing things***: wandb* | *soundfile* | *auraloss* | *audiomentations* | *pedalboard* | *ml_collections* | *omegaconf* | *einops* | *beartype* | *rotary_embedding_torch***

At the end of installing the requirements, it will show this error:

ERROR: ERROR: Failed to build installable wheels for some pyproject.toml based projects (wxpython, diffq, pesq)

There's 2 solutions:

(thanks becruily for the commands!)

if pesq fail - `sudo apt-get install build-essential`

```
pip install numpy==1.26.4
```

Or just remove **wx-python** from *requirements.txt*, because it is used for the GUI only and not for training and “makes requirements install crash on some systems (like colab)” (jarredou)

11 - The End:

Well, I think that mostly covers it. Of course, more experienced people are invited to modify this document as they wish.

I would like to thank:

- Bas Curtiz, for helping me with the metal dataset when I started training and also for his guide about this very topic.
- unwa, for helping me setting up the training repo on the advice of Bas, he has been an incredible help and I hope this guide will help as much as he helped me. Thank you!
- ZFTurbo, for providing the resources to even train a model in the first place and for trusting me with a private model that I am fine-tuning right now! I also want to cite your help for the validation dataset too.
- Kimberley Jensen, for providing the best base roformer and for helping me set up the cloud training process.
- jarredou and becruily for additional help when troubleshooting errors.
- YOU, the newbie reading this document! I hope this helps you to know everything you need about training!