
```python
import os
import logging

logger = logging.getLogger(__name__)

import librosa
import numpy as np
import soundfile as sf
import torch

from infer.lib.uvr5_pack.lib_v5 import nets_61968KB as Nets
from infer.lib.uvr5_pack.lib_v5 import spec_utils
from infer.lib.uvr5_pack.lib_v5.model_param_init import ModelParameters
from infer.lib.uvr5_pack.lib_v5.nets_new import CascadedNet
from infer.lib.uvr5_pack.utils import inference


class AudioPre:
    def __init__(self, agg, model_path, device, is_half, tta=False):
        self.model_path = model_path
        self.device = device
        self.data = {
            # Processing Options
            "postprocess": False,
            "tta": tta,
            # Constants
            "window_size": 512,
            "agg": agg,
            "high_end_process": "mirroring",
        }
        mp = ModelParameters("infer/lib/uvr5_pack/lib_v5/modelparams/4band_v2.json")
        model = Nets.CascadedASPPNet(mp.param["bins"] * 2)
        cpk = torch.load(model_path, map_location="cpu")
        model.load_state_dict(cpk)
        model.eval()
        if is_half:
```

```
            model = model.half().to(device)
        else:
            model = model.to(device)

        self.mp = mp
        self.model = model

    def _path_audio_(
        self, music_file, ins_root=None, vocal_root=None, format="flac", is_hp3=False
    ):
        if ins_root is None and vocal_root is None:
            return "No save root."
        name = os.path.basename(music_file)
        if ins_root is not None:
            os.makedirs(ins_root, exist_ok=True)
        if vocal_root is not None:
            os.makedirs(vocal_root, exist_ok=True)
        X_wave, y_wave, X_spec_s, y_spec_s = {}, {}, {}, {}
        bands_n = len(self.mp.param["band"])
        # print(bands_n)
        for d in range(bands_n, 0, -1):
            bp = self.mp.param["band"][d]
            if d == bands_n:  # high-end band
                (
                    X_wave[d],
                    _,
                ) = librosa.load(  # 理论上librosa读取可能对某些音频有bug，应该上ffmpeg读取，但是太麻烦了
                    music_file,
                    sr=bp["sr"],
                    mono=False,
                    dtype=np.float32,
                    res_type=bp["res_type"],
                )
                if X_wave[d].ndim == 1:
                    X_wave[d] = np.asfortranarray([X_wave[d], X_wave[d]])
            else:  # lower bands
                X_wave[d] = librosa.resample(
                    X_wave[d + 1],
                    orig_sr=self.mp.param["band"][d + 1]["sr"],
```

```
                ]

            X_spec_m = spec_utils.combine_spectrograms(X_spec_s, self.mp)
            aggresive_set = float(self.data["agg"] / 100)
            aggressiveness = {
                "value": aggresive_set,
                "split_bin": self.mp.param["band"][1]["crop_stop"],
            }
            with torch.no_grad():
                pred, X_mag, X_phase = inference(
                    X_spec_m, self.device, self.model, aggressiveness, self.data
                )
            # Postprocess
            if self.data["postprocess"]:
                pred_inv = np.clip(X_mag - pred, 0, np.inf)
                pred = spec_utils.mask_silence(pred, pred_inv)
            y_spec_m = pred * X_phase
            v_spec_m = X_spec_m - y_spec_m

            if ins_root is not None:
                if self.data["high_end_process"].startswith("mirroring"):
                    input_high_end_ = spec_utils.mirroring(
                        self.data["high_end_process"], y_spec_m, input_high_end, self.mp
                    )
                    wav_instrument = spec_utils.cmb_spectrogram_to_wave(
                        y_spec_m, self.mp, input_high_end_h, input_high_end_
                    )
                else:
                    wav_instrument = spec_utils.cmb_spectrogram_to_wave(y_spec_m, self.mp)
                logger.info("%s instruments done" % name)
                if is_hp3 == True:
                    head = "vocal_"
                else:
                    head = "instrument_"
                if format in ["wav", "flac"]:
                    sf.write(
                        os.path.join(
                            ins_root,
                            head + "{}_{}.{}".format(name, self.data["agg"], format),
                        ),
                        (np.array(wav_instrument) * 32768).astype("int16"),
                        self.mp.param["sr"],
                    )  #
                else:
                    path = os.path.join(
                        ins_root, head + "{}_{}.wav".format(name, self.data["agg"])
                    )
                    sf.write(
```

```
                        path,
                        (np.array(wav_instrument) * 32768).astype("int16"),
                        self.mp.param["sr"],
                    )
                    if os.path.exists(path):
                        opt_format_path = path[:-4] + ".%s" % format
                        os.system('ffmpeg -i "%s" -vn "%s" -q:a 2 -y' % (path, opt_format_path))
                        if os.path.exists(opt_format_path):
                            try:
                                os.remove(path)
                            except:
                                pass
            if vocal_root is not None:
                if is_hp3 == True:
                    head = "instrument_"
                else:
                    head = "vocal_"
                if self.data["high_end_process"].startswith("mirroring"):
                    input_high_end_ = spec_utils.mirroring(
                        self.data["high_end_process"], v_spec_m, input_high_end, self.mp
                    )
                    wav_vocals = spec_utils.cmb_spectrogram_to_wave(
                        v_spec_m, self.mp, input_high_end_h, input_high_end_
                    )
                else:
                    wav_vocals = spec_utils.cmb_spectrogram_to_wave(v_spec_m, self.mp)
                logger.info("%s vocals done" % name)
                if format in ["wav", "flac"]:
                    sf.write(
                        os.path.join(
                            vocal_root,
                            head + "{}_{}.{}".format(name, self.data["agg"], format),
                        ),
                        (np.array(wav_vocals) * 32768).astype("int16"),
                        self.mp.param["sr"],
                    )
                else:
                    path = os.path.join(
                        vocal_root, head + "{}_{}.wav".format(name, self.data["agg"])
                    )
                    sf.write(
                        path,
                        (np.array(wav_vocals) * 32768).astype("int16"),
                        self.mp.param["sr"],
                    )
                    if os.path.exists(path):
                        opt_format_path = path[:-4] + ".%s" % format
                        os.system('ffmpeg -i "%s" -vn "%s" -q:a 2 -y' % (path, opt_format_path))
                        if os.path.exists(opt_format_path):
                            try:
                                os.remove(path)
                            except:
                                pass
```

Code Blame Raw

```
            self.model_path = model_path
            self.device = device
            self.data = {
                # Processing Options
                "postprocess": False,
                "tta": tta,
                # Constants
                "window_size": 512,
                "agg": agg,
                "high_end_process": "mirroring",
            }
            mp = ModelParameters("infer/lib/uvr5_pack/lib_v5/modelparams/4band_v3.json")
            nout = 64 if "DeReverb" in model_path else 48
            model = CascadedNet(mp.param["bins"] * 2, nout)
            cpk = torch.load(model_path, map_location="cpu")
            model.load_state_dict(cpk)
            model.eval()
            if is_half:
                model = model.half().to(device)
            else:
                model = model.to(device)

            self.mp = mp
            self.model = model

        def _path_audio_(
            self, music_file, vocal_root=None, ins_root=None, format="flac", is_hp3=False
        ):  # 3个VR模型vocal和ins是反的
            if ins_root is None and vocal_root is None:
                return "No save root."
            name = os.path.basename(music_file)
            if ins_root is not None:
                os.makedirs(ins_root, exist_ok=True)
            if vocal_root is not None:
                os.makedirs(vocal_root, exist_ok=True)
            X_wave, y_wave, X_spec_s, y_spec_s = {}, {}, {}, {}
            bands_n = len(self.mp.param["band"])
            # print(bands_n)
            for d in range(bands_n, 0, -1):
                bp = self.mp.param["band"][d]
                if d == bands_n:  # high-end band
                    (
                        X_wave[d],
                        _,
                    ) = librosa.load(  # 理论上librosa读取可能对某些音频有bug，应该上ffmpeg读取，但是太麻烦了
                        music_file,
                        sr=bp["sr"],
                        mono=False,
                        dtype=np.float32,
```

```
                    dtype=np.float32,
                    res_type=bp["res_type"],
                )
                if X_wave[d].ndim == 1:
                    X_wave[d] = np.asfortranarray([X_wave[d], X_wave[d]])
            else:  # lower bands
                X_wave[d] = librosa.resample(
                    X_wave[d + 1],
                    orig_sr=self.mp.param["band"][d + 1]["sr"],
                    target_sr=bp["sr"],
                    res_type=bp["res_type"],
                )
            # Stft of wave source
            X_spec_s[d] = spec_utils.wave_to_spectrogram_mt(
                X_wave[d],
                bp["hl"],
                bp["n_fft"],
                self.mp.param["mid_side"],
                self.mp.param["mid_side_b2"],
                self.mp.param["reverse"],
            )
            # pdb.set_trace()
            if d == bands_n and self.data["high_end_process"] != "none":
                input_high_end_h = (bp["n_fft"] // 2 - bp["crop_stop"]) + (
                    self.mp.param["pre_filter_stop"] - self.mp.param["pre_filter_start"]
                )
                input_high_end = X_spec_s[d][
                    :, bp["n_fft"] // 2 - input_high_end_h : bp["n_fft"] // 2, :
                ]

        X_spec_m = spec_utils.combine_spectrograms(X_spec_s, self.mp)
        aggresive_set = float(self.data["agg"] / 100)
        aggressiveness = {
            "value": aggresive_set,
            "split_bin": self.mp.param["band"][1]["crop_stop"],
        }
        with torch.no_grad():
            pred, X_mag, X_phase = inference(
                X_spec_m, self.device, self.model, aggressiveness, self.data
            )
        # Postprocess
        if self.data["postprocess"]:
            pred_inv = np.clip(X_mag - pred, 0, np.inf)
            pred = spec_utils.mask_silence(pred, pred_inv)
        y_spec_m = pred * X_phase
        v_spec_m = X_spec_m - y_spec_m

        if ins_root is not None:
            if self.data["high_end_process"].startswith("mirroring"):
                input_high_end_ = spec_utils.mirroring(
                    self.data["high_end_process"], y_spec_m, input_high_end, self.mp
                )
                wav_instrument = spec_utils.cmb_spectrogram_to_wave(
```

```
                    y_spec_m, self.mp, input_high_end_h, input_high_end_
                )
            else:
                wav_instrument = spec_utils.cmb_spectrogram_to_wave(y_spec_m, self.mp)
            logger.info("%s instruments done" % name)
            if format in ["wav", "flac"]:
                sf.write(
                    os.path.join(
                        ins_root,
                        "vocal_{}_{}.{}".format(name, self.data["agg"], format),
                    ),
                    (np.array(wav_instrument) * 32768).astype("int16"),
                    self.mp.param["sr"],
                )  #
            else:
                path = os.path.join(
                    ins_root, "vocal_{}_{}.wav".format(name, self.data["agg"])
                )
                sf.write(
                    path,
                    (np.array(wav_instrument) * 32768).astype("int16"),
                    self.mp.param["sr"],
                )
                if os.path.exists(path):
                    opt_format_path = path[:-4] + ".%s" % format
                    os.system('ffmpeg -i "%s" -vn "%s" -q:a 2 -y' % (path, opt_format_path))
                    if os.path.exists(opt_format_path):
                        try:
                            os.remove(path)
                        except:
                            pass
        if vocal_root is not None:
            if self.data["high_end_process"].startswith("mirroring"):
                input_high_end_ = spec_utils.mirroring(
                    self.data["high_end_process"], v_spec_m, input_high_end, self.mp
                )
                wav_vocals = spec_utils.cmb_spectrogram_to_wave(
                    v_spec_m, self.mp, input_high_end_h, input_high_end_
                )
            else:
                wav_vocals = spec_utils.cmb_spectrogram_to_wave(v_spec_m, self.mp)
            logger.info("%s vocals done" % name)
            if format in ["wav", "flac"]:
                sf.write(
                    os.path.join(
                        vocal_root,
                        "instrument_{}_{}.{}".format(name, self.data["agg"], format),
                    ),
                    (np.array(wav_vocals) * 32768).astype("int16"),
                    self.mp.param["sr"],
                )
            else:
                path = os.path.join(
```

```
                    vocal_root, "instrument_{}_{}.wav".format(name, self.data["agg"])
                )
                sf.write(
                    path,
                    (np.array(wav_vocals) * 32768).astype("int16"),
                    self.mp.param["sr"],
                )
                if os.path.exists(path):
                    opt_format_path = path[:-4] + ".%s" % format
                    os.system('ffmpeg -i "%s" -vn "%s" -q:a 2 -y' % (path, opt_format_path))
                    if os.path.exists(opt_format_path):
                        try:
                            os.remove(path)
                        except:
                            pass
```