

Retrieval-based-Voice-Conversion-WebUI / infer / modules / uvr5 / vr.py  aurexav Allow path name contains special characters

81ca325 · 11 months ago



368 lines (353 l...

```
1  import os
2  import logging
3
4  logger = logging.getLogger(__name__)
5
6  import librosa
7  import numpy as np
8  import soundfile as sf
9  import torch
10
11  from infer.lib.uvr5_pack.lib_v5 import nets_61968KB as Nets
12  from infer.lib.uvr5_pack.lib_v5 import spec_utils
13  from infer.lib.uvr5_pack.lib_v5.model_param_init import ModelParameters
14  from infer.lib.uvr5_pack.lib_v5.nets_new import CascadedNet
15  from infer.lib.uvr5_pack.utils import inference
16
17
18  class AudioPre:
19      def __init__(self, agg, model_path, device, is_half, tta=False):
20          self.model_path = model_path
21          self.device = device
22          self.data = {
23              # Processing Options
24              "postprocess": False,
25              "tta": tta,
26              # Constants
27              "window_size": 512,
28              "agg": agg,
29              "high_end_process": "mirroring",
30          }
31          mp = ModelParameters("infer/lib/uvr5_pack/lib_v5/modelparams/4band_v2.json")
32          model = Nets.CascadedASPPNet(mp.param["bins"] * 2)
33          cpk = torch.load(model_path, map_location="cpu")
34          model.load_state_dict(cpk)
35          model.eval()
36          if is_half:
```

```

37         model = model.half().to(device)
38     else:
39         model = model.to(device)
40
41     self.mp = mp
42     self.model = model
43
44     def _path_audio_(
45         self, music_file, ins_root=None, vocal_root=None, format="flac", is_hp3=False
46     ):
47         if ins_root is None and vocal_root is None:
48             return "No save root."
49         name = os.path.basename(music_file)
50         if ins_root is not None:
51             os.makedirs(ins_root, exist_ok=True)
52         if vocal_root is not None:
53             os.makedirs(vocal_root, exist_ok=True)
54         X_wave, y_wave, X_spec_s, y_spec_s = {}, {}, {}, {}
55         bands_n = len(self.mp.param["band"])
56         # print(bands_n)
57         for d in range(bands_n, 0, -1):
58             bp = self.mp.param["band"][d]
59             if d == bands_n: # high-end band
60                 (
61                     X_wave[d],
62                     _,
63                 ) = 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
64                     music_file,
65                     sr=bp["sr"],
66                     mono=False,
67                     dtype=np.float32,
68                     res_type=bp["res_type"],
69                 )
70             if X_wave[d].ndim == 1:
71                 X_wave[d] = np.asfortranarray([X_wave[d], X_wave[d]])
72         else: # lower bands
73             X_wave[d] = librosa.resample(
74                 X_wave[d + 1],
75                 orig_sr=self.mp.param["band"][d + 1]["sr"],

```

```

95         ]
96
97     X_spec_m = spec_utils.combine_spectrograms(X_spec_s, self.mp)
98     aggressive_set = float(self.data["agg"] / 100)
99     aggressiveness = {
100         "value": aggressive_set,
101         "split_bin": self.mp.param["band"][1]["crop_stop"],
102     }
103     with torch.no_grad():
104         pred, X_mag, X_phase = inference(
105             X_spec_m, self.device, self.model, aggressiveness, self.data
106         )
107     # Postprocess
108     if self.data["postprocess"]:
109         pred_inv = np.clip(X_mag - pred, 0, np.inf)
110         pred = spec_utils.mask_silence(pred, pred_inv)
111     y_spec_m = pred * X_phase
112     v_spec_m = X_spec_m - y_spec_m
113
114     if ins_root is not None:
115         if self.data["high_end_process"].startswith("mirroring"):
116             input_high_end_ = spec_utils.mirroring(
117                 self.data["high_end_process"], y_spec_m, input_high_end, self.mp
118             )
119             wav_instrument = spec_utils.cmb_spectrogram_to_wave(
120                 y_spec_m, self.mp, input_high_end_h, input_high_end_
121             )
122         else:
123             wav_instrument = spec_utils.cmb_spectrogram_to_wave(y_spec_m, self.mp)
124         logger.info("%s instruments done" % name)
125         if is_hp3 == True:
126             head = "vocal_"
127         else:
128             head = "instrument_"
129         if format in ["wav", "flac"]:
130             sf.write(
131                 os.path.join(
132                     ins_root,
133                     head + "{}_{}.{}".format(name, self.data["agg"], format),
134                 ),
135                 (np.array(wav_instrument) * 32768).astype("int16"),
136                 self.mp.param["sr"],
137             ) #
138         else:
139             path = os.path.join(
140                 ins_root, head + "{}_{}.wav".format(name, self.data["agg"])
141             )
142             sf.write(

```

```

143         path,
144         (np.array(wav_instrument) * 32768).astype("int16"),
145         self.mp.param["sr"],
146     )
147     if os.path.exists(path):
148         opt_format_path = path[:-4] + ".%s" % format
149         os.system('ffmpeg -i "%s" -vn "%s" -q:a 2 -y' % (path, opt_format_path))
150         if os.path.exists(opt_format_path):
151             try:
152                 os.remove(path)
153             except:
154                 pass
155     if vocal_root is not None:
156         if is_hp3 == True:
157             head = "instrument_"
158         else:
159             head = "vocal_"
160         if self.data["high_end_process"].startswith("mirroring"):
161             input_high_end_ = spec_utils.mirroring(
162                 self.data["high_end_process"], v_spec_m, input_high_end, self.mp
163             )
164             wav_vocals = spec_utils.cmb_spectrogram_to_wave(
165                 v_spec_m, self.mp, input_high_end_h, input_high_end_
166             )
167         else:
168             wav_vocals = spec_utils.cmb_spectrogram_to_wave(v_spec_m, self.mp)
169         logger.info("%s vocals done" % name)
170         if format in ["wav", "flac"]:
171             sf.write(
172                 os.path.join(
173                     vocal_root,
174                     head + "{}_{}.{}".format(name, self.data["agg"], format),
175                 ),
176                 (np.array(wav_vocals) * 32768).astype("int16"),
177                 self.mp.param["sr"],
178             )
179         else:
180             path = os.path.join(
181                 vocal_root, head + "{}_{}.wav".format(name, self.data["agg"])
182             )
183             sf.write(
184                 path,
185                 (np.array(wav_vocals) * 32768).astype("int16"),
186                 self.mp.param["sr"],
187             )
188             if os.path.exists(path):
189                 opt_format_path = path[:-4] + ".%s" % format
190                 os.system('ffmpeg -i "%s" -vn "%s" -q:a 2 -y' % (path, opt_format_path))
191                 if os.path.exists(opt_format_path):
192                     try:
193                         os.remove(path)
194                     except:
195                         pass

```

Code

Blame

Raw



```

200     self.model_path = model_path
201     self.device = device
202     self.data = {
203         # Processing Options
204         "postprocess": False,
205         "tta": tta,
206         # Constants
207         "window_size": 512,
208         "agg": agg,
209         "high_end_process": "mirroring",
210     }
211     mp = ModelParameters("infer/lib/uvr5_pack/lib_v5/modelparams/4band_v3.json")
212     nout = 64 if "DeReverb" in model_path else 48
213     model = CascadedNet(mp.param["bins"] * 2, nout)
214     cpk = torch.load(model_path, map_location="cpu")
215     model.load_state_dict(cpk)
216     model.eval()
217     if is_half:
218         model = model.half().to(device)
219     else:
220         model = model.to(device)
221
222     self.mp = mp
223     self.model = model
224
225     def _path_audio_(
226         self, music_file, vocal_root=None, ins_root=None, format="flac", is_hp3=False
227     ): # 3个VR模型vocal和ins是反的
228         if ins_root is None and vocal_root is None:
229             return "No save root."
230         name = os.path.basename(music_file)
231         if ins_root is not None:
232             os.makedirs(ins_root, exist_ok=True)
233         if vocal_root is not None:
234             os.makedirs(vocal_root, exist_ok=True)
235         X_wave, y_wave, X_spec_s, y_spec_s = {}, {}, {}, {}
236         bands_n = len(self.mp.param["band"])
237         # print(bands_n)
238         for d in range(bands_n, 0, -1):
239             bp = self.mp.param["band"][d]
240             if d == bands_n: # high-end band
241                 (
242                     X_wave[d],
243                     y_wave[d],
244                 ) = 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
245                     music_file,
246                     sr=bp["sr"],
247                     mono=False,
248                     dtype=np.float32

```

```

243         dtype=np.float32,
244         res_type=bp["res_type"],
245     )
246     if X_wave[d].ndim == 1:
247         X_wave[d] = np.asfortranarray([X_wave[d], X_wave[d]])
248     else: # lower bands
249         X_wave[d] = librosa.resample(
250             X_wave[d + 1],
251             orig_sr=self.mp.param["band"][d + 1]["sr"],
252             target_sr=bp["sr"],
253             res_type=bp["res_type"],
254         )
255     # Stft of wave source
256     X_spec_s[d] = spec_utils.wave_to_spectrogram_mt(
257         X_wave[d],
258         bp["hl"],
259         bp["n_fft"],
260         self.mp.param["mid_side"],
261         self.mp.param["mid_side_b2"],
262         self.mp.param["reverse"],
263     )
264     # pdb.set_trace()
265     if d == bands_n and self.data["high_end_process"] != "none":
266         input_high_end_h = (bp["n_fft"] // 2 - bp["crop_stop"]) + (
267             self.mp.param["pre_filter_stop"] - self.mp.param["pre_filter_start"]
268         )
269         input_high_end = X_spec_s[d][
270             :, bp["n_fft"] // 2 - input_high_end_h : bp["n_fft"] // 2, :
271         ]
272
273     X_spec_m = spec_utils.combine_spectrograms(X_spec_s, self.mp)
274     aggressive_set = float(self.data["agg"] / 100)
275     aggressiveness = {
276         "value": aggressive_set,
277         "split_bin": self.mp.param["band"][1]["crop_stop"],
278     }
279     with torch.no_grad():
280         pred, X_mag, X_phase = inference(
281             X_spec_m, self.device, self.model, aggressiveness, self.data
282         )
283     # Postprocess
284     if self.data["postprocess"]:
285         pred_inv = np.clip(X_mag - pred, 0, np.inf)
286         pred = spec_utils.mask_silence(pred, pred_inv)
287     y_spec_m = pred * X_phase
288     v_spec_m = X_spec_m - y_spec_m
289
290     if ins_root is not None:
291         if self.data["high_end_process"].startswith("mirroring"):
292             input_high_end_ = spec_utils.mirroring(
293                 self.data["high_end_process"], y_spec_m, input_high_end, self.mp
294             )
295             wav_instrument = spec_utils.cmb_spectrogram_to_wave(

```

```

301         y_spec_m, self.mp, input_high_end_h, input_high_end_
302     )
303     else:
304         wav_instrument = spec_utils.cmb_spectrogram_to_wave(y_spec_m, self.mp)
305         logger.info("%s instruments done" % name)
306         if format in ["wav", "flac"]:
307             sf.write(
308                 os.path.join(
309                     ins_root,
310                     "vocal_{}_{}.{}".format(name, self.data["agg"], format),
311                 ),
312                 (np.array(wav_instrument) * 32768).astype("int16"),
313                 self.mp.param["sr"],
314             ) #
315         else:
316             path = os.path.join(
317                 ins_root, "vocal_{}_{}.wav".format(name, self.data["agg"])
318             )
319             sf.write(
320                 path,
321                 (np.array(wav_instrument) * 32768).astype("int16"),
322                 self.mp.param["sr"],
323             )
324             if os.path.exists(path):
325                 opt_format_path = path[:-4] + ".%s" % format
326                 os.system('ffmpeg -i "%s" -vn "%s" -q:a 2 -y' % (path, opt_format_path))
327                 if os.path.exists(opt_format_path):
328                     try:
329                         os.remove(path)
330                     except:
331                         pass
332             if vocal_root is not None:
333                 if self.data["high_end_process"].startswith("mirroring"):
334                     input_high_end_ = spec_utils.mirroring(
335                         self.data["high_end_process"], v_spec_m, input_high_end, self.mp
336                     )
337                     wav_vocals = spec_utils.cmb_spectrogram_to_wave(
338                         v_spec_m, self.mp, input_high_end_h, input_high_end_
339                     )
340                 else:
341                     wav_vocals = spec_utils.cmb_spectrogram_to_wave(v_spec_m, self.mp)
342                     logger.info("%s vocals done" % name)
343                     if format in ["wav", "flac"]:
344                         sf.write(
345                             os.path.join(
346                                 vocal_root,
347                                 "instrument_{}_{}.{}".format(name, self.data["agg"], format),
348                             ),
349                             (np.array(wav_vocals) * 32768).astype("int16"),
350                             self.mp.param["sr"],
351                         )
352                     else:
353                         path = os.path.join(

```

```
354         vocal_root, "instrument_{}_{}.wav".format(name, self.data["agg"])
355     )
356     sf.write(
357         path,
358         (np.array(wav_vocals) * 32768).astype("int16"),
359         self.mp.param["sr"],
360     )
361     if os.path.exists(path):
362         opt_format_path = path[:-4] + ".%s" % format
363         os.system('ffmpeg -i "%s" -vn "%s" -q:a 2 -y' % (path, opt_format_path))
364         if os.path.exists(opt_format_path):
365             try:
366                 os.remove(path)
367             except:
368                 pass
```