PyAudioCensor

Name : Mayukhmali Das
Degree: Bachelor of Engineering
Discipline: Electronics and Telecommunications
Year of College: 4th(Final)
College Name: Jadavpur University
Country: India
Website: https://smartmayukh.github.io/Personal-site/
Github: https://github.com/Smartmayukh

My own Python Library Implementation (PyAudioCensor)

I created my own python package named PyAudioCensor. Firstly we have to give the audio file to be censored. Then we perform offline speech recognition on the audio using Vosk api. Thus we will get a list of words with their respective timestamps. Now the user will input which words to censor. The respective timestamps of those words are found out and then the censor audio provided by the user is overlaid at those instances.

Audio files can be automatically censored using the censor_audio function of my PyAudioCensor module. Process will be completely offline.

```
def censor_audio(base_audio_path, censor_audio_path, output_audio_path,
model_path, to_censor, gain_of_censor=0, gain_of_base=-40,  silent=1):
```

Description of function parameters:

1. **Base_Audio_Path:** The path to the base audio to censor.

2. **Censor_Audio_Path:** The path to the audio which will be used to censor.

3. **Output_Audio_Path:** The location where the censored audio will be stored.

4. **Model_path:** Path for the Vosk Model
   https://alphacephei.com/vosk/models

   Vosk API is a speech recognition toolkit developed by Alpha Cephei Inc., which is based on the Kaldi toolkit. Kaldi is a free and open-source toolkit for speech recognition developed by the Johns Hopkins University Speech and Language Processing Group. It uses modern machine learning techniques, including deep neural networks, to achieve high accuracy and efficiency.

   The **offline speech recognition with timestamps part** is done with the help of Dmytro Nikolaiev work referenced below:
   https://gitlab.com/Winston-90/foreign_speech_recognition/-/tree/main/timestamps

5. **To_censor:** List of words to be censored

6. **Gain_of_Base :** Gain of the base audio.

7. **Gain_of_Overlay:** Gain of the censor audio.

8. **Silent :** Make base audio silent while overlaying the censor audio.

Full Code:

```
import wave
import json
from vosk import Model, KaldiRecognizer, SetLogLevel

class Word:


    def __init__(self, dict):


        self.conf = dict["conf"]
        self.end = dict["end"]
        self.start = dict["start"]
        self.word = dict["word"]

    def to_string(self):

        return "{:20} from {:.2f} sec to {:.2f} sec, confidence is
{:.2f}%".format(
            self.word, self.start, self.end, self.conf*100)


    # def start_pointt(self):

    #     return "{:20},{:.2f}".format(
    #         self.word, self.start)

    def start_point(self):

        return [self.word, self.start]



def timestamp_list (base_audio_path,model_path):

    audio_filename = base_audio_path
```

```python
model = Model("PyAudioCensor\model")
wf = wave.open(audio_filename, "rb")
rec = KaldiRecognizer(model, wf.getframerate())
rec.SetWords(True)

# get the list of JSON dictionaries
results = []
# recognize speech using vosk model
while True:
    data = wf.readframes(4000)
    if len(data) == 0:
        break
    if rec.AcceptWaveform(data):
        part_result = json.loads(rec.Result())
        results.append(part_result)
part_result = json.loads(rec.FinalResult())
results.append(part_result)

# convert list of JSON dictionaries to list of 'Word' objects
list_of_Words = []
for sentence in results:
    if len(sentence) == 1:
        # sometimes there are bugs in recognition
        # and it returns an empty dictionary
        # {'text': ''}
        continue
    for obj in sentence['result']:
        w = Word(obj)  # create custom Word object
        list_of_Words.append(w)  # and add it to list

wf.close()  # close audiofile

final=[]

# output to the screen
for word in list_of_Words:
    print(word.to_string())
```

```
    print("Now taking only the word and its starting time of occurence")

    for word in list_of_Words:
        time = word.start_point()
        final.append(time)

    for value in final:
        print(value)

    return final



def censor_audio(base_audio_path, censor_audio_path, output_audio_path,
model_path, to_censor, gain_of_censor=0, gain_of_base=-40,  silent=1):

    time_list=timestamp_list(base_audio_path,model_path)

    def find_time_occurrences(to_censor):
        result = []
        for word in to_censor:
            for item in time_list:
                if item[0] == word:
                    result.append(item[1]*1000)
        return result

    censor_time=find_time_occurrences(to_censor)

    positions=censor_time

    for value in censor_time:
       print(value)

    # Open the base audio file
    base_audio_file = wave.open(base_audio_path, "rb")
    base_audio_params = base_audio_file.getparams()
    base_audio_frames = base_audio_file.readframes(base_audio_params.nframes)
```

```
    # Open the censor audio file
    censor_audio_file = wave.open(censor_audio_path, "rb")
    censor_audio_params = censor_audio_file.getparams()
    censor_audio_frames =
censor_audio_file.readframes(censor_audio_params.nframes)

    # Define a function to convert dB to float
    def db_to_float(db):
        return 10 ** (db / 10)

    # Convert the audio frames to integers
    base_samples = list(base_audio_frames)
    censor_samples = list(censor_audio_frames)

    # Define the gain during censor
    base_gain = db_to_float(gain_of_base)

    # Apply gain to the censor audio if necessary
    if gain_of_censor is not None:
        # Convert gain from dB to float
        censor_gain = db_to_float(gain_of_censor)
        # Apply gain to the censor samples
        censor_samples = [int(sample * censor_gain) for sample in censor_samples]

    # Iterate over each position in the positions list
    for position in positions:
        # Calculate the position in samples
        position_samples = int(position / 500.0 * base_audio_params.framerate)

        # Insert the censor audio at the desired position
        for i in range(len(censor_samples)):
            if silent == 1:  # Check if silent mode is enabled
                base_samples[position_samples + i] = censor_samples[i]
            else:
                base_samples[position_samples + i] =
int(base_samples[position_samples + i] * base_gain) + censor_samples[i]
            # Clip the values to the valid range of 0 to 255
            base_samples[position_samples + i] = max(0,
min(base_samples[position_samples + i], 255))
```

```python
    # Save the mixed audio as a new file
    with wave.open(output_audio_path, "wb") as mixed_file:
        mixed_file.setparams(base_audio_params)
        mixed_file.writeframes(bytearray(base_samples))
    return(output_audio_path)
```

Example of implementation :

I will be demonstrating the working of this package using the song "Happier" by Marshmello

```python
from PyAudioCensor import main

main.censor_audio("base_audio.wav","overlay_audio.wav"
,"censored.wav",model_path="PyCensorAudio\model",
to_censor=["happier","morning","story","mind"],silent=
1)
```

Here is the speech recognition of the song along with timestamps given by Vosk

they from 0.33 sec to 0.62 sec, confidence is 63.98%
they from 0.62 sec to 1.23 sec, confidence is 100.00%
are from 1.26 sec to 1.59 sec, confidence is 100.00%
the from 1.59 sec to 2.13 sec, confidence is 100.00%
oven from 2.16 sec to 2.76 sec, confidence is 30.85%
thing from 2.76 sec to 3.60 sec, confidence is 100.00%
i from 3.63 sec to 3.96 sec, confidence is 100.00%
want from 3.96 sec to 4.32 sec, confidence is 100.00%
you from 4.32 sec to 4.53 sec, confidence is 100.00%
to from 4.53 sec to 4.83 sec, confidence is 100.00%
be from 4.86 sec to 5.13 sec, confidence is 100.00%
happier from 5.13 sec to 5.97 sec, confidence is 84.00%
i from 6.03 sec to 6.33 sec, confidence is 100.00%
want from 6.33 sec to 6.72 sec, confidence is 100.00%
you from 6.72 sec to 6.90 sec, confidence is 100.00%
to from 6.90 sec to 7.20 sec, confidence is 100.00%
be from 7.20 sec to 7.53 sec, confidence is 100.00%
happier from 7.53 sec to 8.94 sec, confidence is 100.00%
when from 8.97 sec to 9.21 sec, confidence is 100.00%
the from 9.21 sec to 9.30 sec, confidence is 100.00%
morning from 9.30 sec to 9.90 sec, confidence is 100.00%
comes from 9.90 sec to 10.83 sec, confidence is 100.00%
we from 10.93 sec to 11.10 sec, confidence is 92.56%
see from 11.10 sec to 11.40 sec, confidence is 100.00%
what from 11.40 sec to 11.67 sec, confidence is 98.79%
we've from 11.73 sec to 12.06 sec, confidence is 89.67%
become from 12.06 sec to 12.90 sec, confidence is 83.34%
in from 13.14 sec to 13.47 sec, confidence is 100.00%
colada from 13.50 sec to 14.10 sec, confidence is 100.00%
day from 14.13 sec to 14.37 sec, confidence is 93.45%
where from 14.37 sec to 14.63 sec, confidence is 68.65%
flaming from 14.67 sec to 15.33 sec, confidence is 97.49%
when from 15.33 sec to 15.57 sec, confidence is 90.09%
notified from 15.57 sec to 16.44 sec, confidence is 100.00%
we from 16.47 sec to 16.83 sec, confidence is 90.59%
be from 16.83 sec to 17.49 sec, confidence is 100.00%
gone from 17.49 sec to 18.33 sec, confidence is 100.00%
every from 18.60 sec to 18.96 sec, confidence is 100.00%
argument from 18.96 sec to 20.10 sec, confidence is 100.00%
every from 20.43 sec to 20.76 sec, confidence is 100.00%

```
word            from 20.76 sec to 21.09 sec, confidence is 100.00%
we              from 21.09 sec to 21.27 sec, confidence is 100.00%
translate       from 21.30 sec to 21.93 sec, confidence is 100.00%
back            from 21.93 sec to 22.71 sec, confidence is 100.00%
just            from 22.77 sec to 23.01 sec, confidence is 77.98%
read            from 23.01 sec to 23.22 sec, confidence is 39.86%
all             from 23.22 sec to 23.37 sec, confidence is 77.98%
that            from 23.37 sec to 23.58 sec, confidence is 100.00%
has             from 23.58 sec to 23.76 sec, confidence is 100.00%
happened        from 23.76 sec to 24.12 sec, confidence is 100.00%
nothing         from 24.12 sec to 24.57 sec, confidence is 100.00%
now             from 24.57 sec to 24.72 sec, confidence is 47.76%
we              from 24.72 sec to 24.90 sec, confidence is 57.61%
prefer          from 24.90 sec to 25.35 sec, confidence is 57.61%
to              from 25.35 sec to 25.50 sec, confidence is 56.25%
play            from 25.50 sec to 25.74 sec, confidence is 56.25%
the             from 25.74 sec to 25.98 sec, confidence is 45.44%
story           from 25.98 sec to 27.39 sec, confidence is 100.00%
i               from 32.43 sec to 32.67 sec, confidence is 41.46%
wanted          from 32.68 sec to 33.16 sec, confidence is 29.49%
to              from 33.16 sec to 33.24 sec, confidence is 43.02%
change          from 33.25 sec to 33.66 sec, confidence is 76.04%
my              from 33.66 sec to 33.94 sec, confidence is 81.84%
mind            from 33.94 sec to 34.50 sec, confidence is 73.78%
right           from 36.33 sec to 36.90 sec, confidence is 100.00%
the             from 47.16 sec to 47.61 sec, confidence is 100.00%
hey             from 48.72 sec to 57.06 sec, confidence is 26.46%
```

Our interest is in the words and their respective starting times. So we format the above output as below:

```
['they', 0.33]
['they', 0.621496]
['are', 1.26]
['the', 1.59]
['oven', 2.16]
['thing', 2.76]
['i', 3.63]
['want', 3.96]
['you', 4.32]
['to', 4.53]
```

['be', 4.86]
['happier', 5.13]
['i', 6.03]
['want', 6.33]
['you', 6.72]
['to', 6.9]
['be', 7.2]
['happier', 7.53]
['when', 8.97]
['the', 9.21]
['morning', 9.3]
['comes', 9.9]
['we', 10.933383]
['see', 11.1]
['what', 11.4]
["we've", 11.73]
['become', 12.06]
['in', 13.14]
['colada', 13.5]
['day', 14.13]
['where', 14.37]
['flaming', 14.67]
['when', 15.33]
['notified', 15.569462]
['we', 16.47]
['gone', 17.49]
['every', 18.6]
['argument', 18.96]
['every', 20.43]
['word', 20.76]
['we', 21.09]
['translate', 21.3]
['back', 21.93]
['just', 22.77]
['read', 23.01]
['all', 23.22]
['that', 23.37]
['has', 23.58]
['happened', 23.76]
['nothing', 24.12]
['now', 24.57]
['we', 24.72]
['prefer', 24.9]
['to', 25.35]

['play', 25.5]
['the', 25.74]
['story', 25.977642]
['i', 32.43]
['wanted', 32.67538]
['to', 33.157621]
['change', 33.24668]
['my', 33.66]
['mind', 33.944718]
['right', 36.33]
['the', 47.16]
['hey', 48.72]

Now the user will have to give a list of words to be censored. The below code will find all the occurrences of those words and will create a list of starting times to be censored.

```
time_list=timestamp_list(base_audio_path,model_path)

    def find_time_occurrences(to_censor):
        result = []
        for word in to_censor:
            for item in time_list:
                if item[0] == word:
                    result.append(item[1]*1000)
        return result

    censor_time=find_time_occurrences(to_censor)

    positions=censor_time

    for value in censor_time:
       print(value)
```

Here in this case we are censoring the words

["happier","morning","story","mind"].

Below is the list of start times for all these words generated by the above function.

5130.0
7530.0
9300.0
33944.718

Now the censor audio will be overlayed at these particular timestamps(in ms).

Here are links to all three audio files and the full package:

Base Audio

https://drive.google.com/file/d/1A_V6ycl4DMyGEyzNZmv60ZxUZObwIUiL/view?usp=sharing

Overlay Audio

https://drive.google.com/file/d/1blOtMAF08ZCaXR0dIJiqWXnVMI_yipFm/view?usp=share_link

Censored Audio

https://drive.google.com/file/d/1JK9dBF96_Vqx6-LHC6fdkbeyUCqhJnv9/view?usp=sharing

PyAudioCensor Package:

https://drive.google.com/drive/folders/1JXyV5dfNGy_3aUsMrsdOoEzWWJ9vOqTn