VoiceSplit

Targeted Voice Separation by Speaker Conditioned Spectrogram

Partial Report

SCC0251/5830 Image Processing

Edresson Casanova - 11572715
Pedro Regattieri Rocha - 8531702

1. Introduction

VoiceSplit's goal is the development of a system that, given an audio input, is able to separate overlapping voices through the use of Mel Spectrograms, based on the characteristics of each speaker's speech patterns. With this the system will be able to separate people having a conversation into different entries to help populate data sets.

VoiceSplit's neural architecture is based on VoiceFilter: Targeted Voice Separation by Speaker-Conditioned Spectrogram Masking, a paper proposed by Google researchers. This paper uses the Librispeech data set, which will also be used for this project, along the Speech2Phone and Voxceleb 1 & 2 data sets, also used in the Google paper.

The initial step is the use of the normalise-resample.sh script to convert .flac lossless sound files into .wav, single channel mono sound files that will be used by VoiceSplit.

The process for data processing is the same as outlined in VoiceFilter, using the preprocess_by_csv.py script that reads .csv files and creates a data set from it using the pandas library.

After that, we remove the initial and final silence from the files, and, as is the case for VoiceFilter, only three seconds of each audio is considered. Furthermore, the audio in these three seconds contains the overlapping voices of the speakers in the file.

The expected audio, the predicted audio and the embedding reference are then saved. VoiceSplit supports three different embedding systems: GE2E CorentinJ, a model trained with the Librispeech and both Voxceleb data sets; GE2E Seungwonpark, used during our initial training, which was trained using only Voxceleb 2 and Speech2Phone, which contains one hour of audio featuring 40 different speakers.

We used MSE as our loss function so we could start training our models as soon as possible, as training time is the bottleneck of our project.

We had access to a V100 machine for training purposes, allocated to us for one week. However, due to issues with data pre-processing, the long time needed to train each model, and the limited time frame we had access to the V100, ultimately only two days worth of training were performed using the V100 before we had to downgrade to a Nvidia GTX 1080, slowing down the training process.

Once the implementation of Powerlaw Compression Loss function is complete, it will replace MSE as the loss function in further training.

Using the V100, we were able to perform 820 thousand steps with a batch size of 24, for a total of 19.680.000 iterations of our training.

2. Sample Images of the Spectrogram Processing

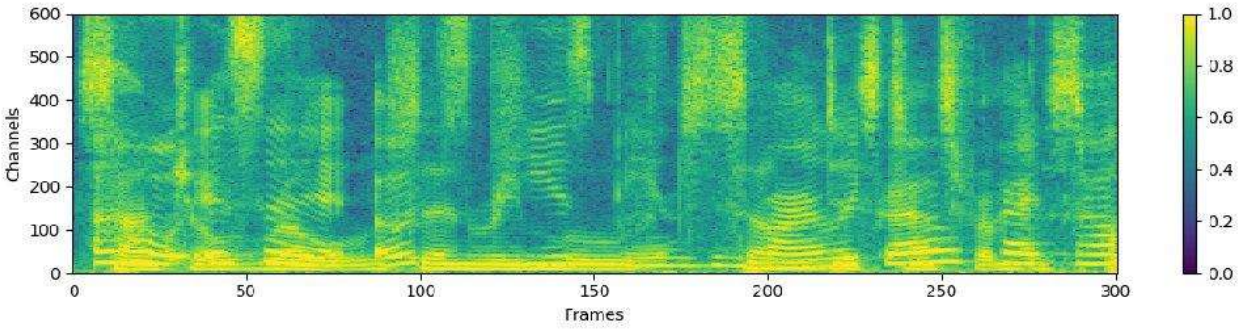

Figure 1: Processed .wav file spectrogram, with overlapping speakers

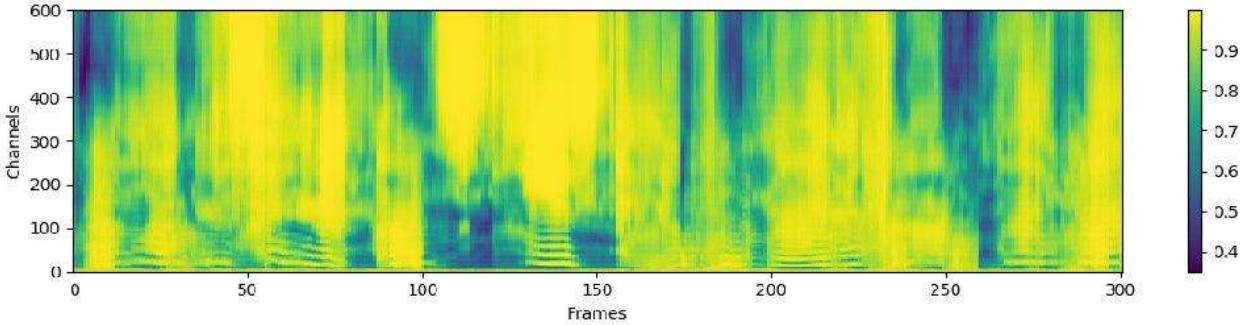

Figure 2: Predicted Mask spectrogram, used to separate the speakers.

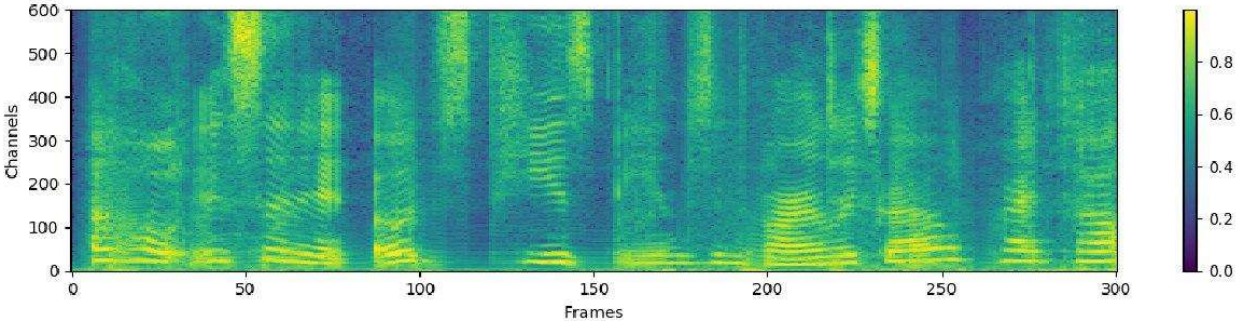

Figure 3: Predicted Audio spectrogram, a single speaker with minor interference from the other. Obtained by multiplying the overlapping speaker spectrogram by the predicted mask spectrogram.

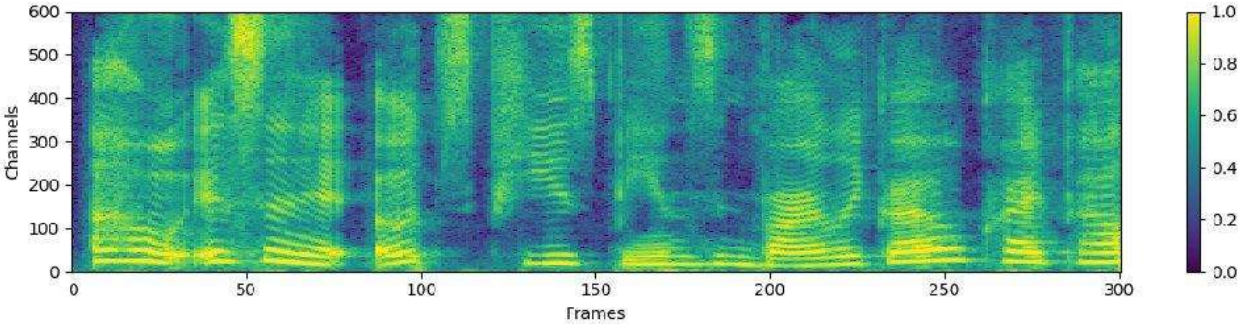

Figure 4: Expected result spectrogram, the spectrogram of a sound file that only has a single speaker, with no interference from another speaker, used as a baseline.

3. Sample Code

The following sample code, a part of preprocess_by_csv.py, reads the .csv files used in the Google paper and creates a data set of .wav files with overlapping speakers. The spectrograms of these .wav files are extracted according to parameters present in the config.json file.

```
if __name__ ==
'__main__':
                def train_wrapper(num):
                clean_utterance_path, embedding_utterance_path, interference_utterance_path =
                train_data[num]
                mix_wavfiles(output_dir_train, sample_rate, audio_len, ap, form, num,
                embedding_utterance_path, interference_utterance_path, clean_utterance_path)
                def test_wrapper(num):
                clean_utterance_path, embedding_utterance_path, interference_utterance_path =
                test_data[num]
                mix_wavfiles(output_dir_test, sample_rate, audio_len, ap, form, num,
                embedding_utterance_path, interference_utterance_path, clean_utterance_path)
                parser = argparse.ArgumentParser()
                parser.add_argument('-c', '--config', type=str, required=True,
                help="Config json file")
                parser.add_argument('-r', '--dataset_root_dir', type=str, required=True,
                help="Config json file")
                parser.add_argument('-d', '--train_data_csv', type=str, required=True,
                help="Train Data csv contains rows
                [clean_utterance,embedding_utterance,interference_utterance] example in
                datasets/LibriSpeech/train.csv")
                parser.add_argument('-t', '--test_data_csv', type=str, required=True,
                help="Test Data csv contains rows
                [clean_utterance,embedding_utterance,interference_utterance] example in
                datasets/LibriSpeech/dev.csv")
                parser.add_argument('-o', '--out_dir', type=str, required=True,
                help="Directory of output training triplet")
                parser.add_argument('-l', '--librispeech', type=str, required=False,
                default=False,
                help="Librispeech format, if true load with librispeech format")
                args = parser.parse_args()
                os.makedirs(args.out_dir, exist_ok=True)
                os.makedirs(os.path.join(args.out_dir, 'train'), exist_ok=True)
                os.makedirs(os.path.join(args.out_dir, 'test'), exist_ok=True)
                cpu_num = cpu_count() # num threads = num cpu cores
                config = load_config(args.config)
                ap = AudioProcessor(config.audio)
                sample_rate = config.audio[config.audio['backend']]['sample_rate']
                audio_len = config.audio['audio_len']
```

```
form = config.dataset['format']
output_dir_train = os.path.join(args.out_dir, 'train')
output_dir_test = os.path.join(args.out_dir, 'test')
dataset_root_dir = args.dataset_root_dir

train_data_csv = pd.read_csv(args.train_data_csv, sep=',').values
test_data_csv = pd.read_csv(args.test_data_csv, sep=',').values
train_data = []
test_data = []
```