# Advanced Audio Processing: Exercise 06

# Singing voice separation with recurrent neural network based denoising auto-encoders

## 1 Introduction

In the previous exercises you have created deep neural networks (DNNs) to do classification and detection. But, classification and detection are only a fraction of the types of tasks that are associated with audio processing. Probably the rest of the tasks can be grouped under the "regression" term. One of the most famous, regression-based audio processing tasks is music source separation. That is, given a mixture of music sources, you have to separate one of them (check the slides of the lectures of the course, for in depth details!). One example is singing voice separation (i.e. separate the singing voice from the rest of the mixture, i.e. the song). In this exercise you will use existing code and develop new types of DNNs focused in a regression task, and specifically in the task of singing voice source separation. You will create a recurrent neural network (RNN) based denoising auto-encoder (DAE) for performing singing voice separation, using a subset of the freely available music source separation dataset DSD100 [1], and evaluate the performance of the separation. In all tasks, you are free to use either a CPU or a graphics processing unit (GPU) for implementing the necessary calculations.

In this exercise we will use an extra python library calledMIR Eval [2] for objectively evaluating the quality of the separation. The rest of the document is organized as follows. In Section 2 you will set up your data, creating your examples from the raw audio data. In Section 3 you will create your RNN-based DAE and use it to do the singing voice separation, by directly predicting the magnitude spectrum of the source, and evaluate the separated signal.

## 2 Setting-up the data (1 point)

In this exercise, you will be given the raw audio data and you will have to create the training and testing examples (no validation, just for demo purposes of this exercise) your-selves. You are free to use existing code (either yours or provided) and external libraries (e.g. librosa). You will have to download the data, perform feature extraction, and set-up your dataset class and data loader in PyTorch.

### 2.1 Download the data and perform feature extraction (0.6 points)

To use the data, you will first have to download the data from the Moodle page of the exercise and then do feature extraction. The provided data are split in two directories, Mixtures and Sources. Each directory has two sub-directories, training and testing. You will have to read the audio files from the directories, extract features, and create the same directory structure that will host your features.

You will have to:

1. Download and expand the dataset in the root directory of your project for this exercise.
2. In each of Mixtures and Sources directories, create two sub-directories, training_features and testing_features.
3. Read every audio file, and extract the short-time Fourier transform (STFT) using 2048 points with hop size 1024 and Hamming window, and the librosa.stft function [3].
4. Split the obtained STFT of each file in sequences of 60 vectors, discarding any extra vectors.
5. Serialize the resulting sequences of vectors using meaningful naming, e.g. **mix_<song_name>_seq_01.npy** for the first sequence of the song with name <song_name>. Save each split in the proper directory e.g. Mixtures for mixtures and in training_features for the training mixtures, the resulted path will be something like e.g. **Mixtures/training_features/mix_<song_name>_seq_01.npy**.

The code that implements the above should be placed in a file called **feature_extraction_<your name>.py**.

### 2.2 Dataset class and data loader (0.4 points)

To efficiently use the above data with PyTorch, you will have to use a sub-class of the Dataset class along with the DataLoader of PyTorch.

The steps to follow are:

1. Create a dataset class, which will accept an indication if it is training or testing split, and the paths of the Mixtures (targeted outputs) and Sources (inputs) directories.
2. According to the above mentioned flag and paths, the dataset class will load the sequences and create corresponding input and output pairs, matching the files under Mixtures with the files under Sources. For example, the file **mix_<song_name>_seq_01.npy** should be the targeted output of the file **sou_<song_name>_seq_01.npy**.
3. In the case of training split, make your dataset sub-class to return the proper examples when asked, as you have done in the previous exercises.
4. In the case of testing split, make your dataset sub-class to return all the vectors of the mixture (**only**) for the specific song, plus the whole raw audio file of the corresponding source and the whole audio of the separated voice. That is, in the case of the testing split the dataset should return all the sequences of the <song_name> song, plus the raw audio file with the <song_name> from the Mixtures directory. This is needed for converting your predicted voice back to audio. Remember that the use case of the testing split is to provide examples for which you have no ground truth values, i.e. you have no separated voice. The reasons that you provide the whole audio files are: i) to be able to retrieve the phase of the mixture, in order to obtain the audio back from the predicted magnitude spectrum, and ii) to be able to do a subjective (or objective in a real-case scenario) evaluation. To reconstruct the signal you can make use of the helper function to_audio() in **exercise_06.py.**

The code implementing the above should be placed in a file called **data_handling_<your name>.py**.

## 3 Implementing and optimizing a recurrent neural network based denoising auto-encoder, and evaluating the output (2 points)

### 3.1 Implementing the DAE (1 point)

The typical characteristic of a denoising auto-encoder (DAE) is that the loss used to optimize the parameters is based on the prediction of the cleaned version of the input. That is, the input to a DAE is a corrupted version of a signal, and the targeted output is the cleaned (un-corrupted) version of it. In the case of singing voice separation, you can consider the singing voice being your clean signal and the mixture its corrupted version. To create your DAE you will use three stacked bi-directional gated recurrent units (GRUs), a uni-directional GRU, and a trainable affine transform with bias (i.e. a feed-forward layer).

The steps to follow are:

- Create a sub-class of the torch.nn.Module that will be your DAE.
- Populate your DAE by adding one bi-directional (Bi-GRU) layer having 1025 input features and 128 output features. The output of the Bi-GRU will be the concatenation of the two different directions.
- Add two more Bi-GRU layers having 256 input and 128 output features. Again, the output of each Bi-GRU layer would be the concatenation of both directions.
- Add one GRU layer having 256 input and 128 output features.
- Add one linear layer having 128 input and 1025 output features, with rectified linear unit as an output.
- Create some dummy data and test if your code has any bugs.

The code implementing the above should be placed in a file called **dae_<your_name>.py**.

### 3.2 Implementing training, validation, and testing process (0.6 points)

Similar to what you have implemented up to now and in the previous exercises, you have to optimize your DAE and, afterwards, test it.

Specifically, you have to:

1. Set-up the training, using 200 as maximum amount of epochs, select MSE loss and implement the testing processes.
2. Use the training split of your data to optimize your DAE.
3. After the training is over, use the testing data to do the testing of your DAE.
4. Use the provided function to convert your predicted spectrum (i.e the output of the testing process) back to audio.
5. Using librosa.output.write wav, save each separated and corresponding ground truth audio file (i.e the output of the provided function for converting the predicted spectrum) at your hard disk.

The code implementing the above should be placed in a file called **mss_<your_name>.py**.

## 3.3 Evaluation of the separation process (0.4 points)

After you have obtained the separated and corresponding ground truth audio files, you can objectively and subjectively evaluate five predicted audio files. To subjectively evaluate them, you can directly listen to them by using an audio player from the computer that you are using. To objectively evaluate them, you have to use the mir_eval.separation.bss_eval_images framewise function [4], with window equal to one second and zero overlap.
Specifically, you have to do:

1. Select five pairs of predicted and ground truth audio, and listen to each pair.
2. Use the predicted and ground truth audio signals as an input to the mir_eval.separation.bss_eval_images_framewise and obtain the values of signal to interference ratio (SIR), signal to artifacts ratio (SAR), and signal to distortion ratio (SDR).
3. Report the values of SIR, SAR, and SDR along with your subjective evaluation.

Write your answers (in comment format) and your code in the file **mss_<your_name>.py**.

Finally, submit for the assessment of this exercise, the following files:
**feature_extraction_<your name>.py, data_handling_<your name>.py, dae_<your_name>.py** and **mss_<your_name>.py** .

If you want to see some real-case scenarios where the above metrics are used (*and listen to the results as well*), you can see the results of the signal separation evaluation campaign (SiSEC) online [5]

**References**
[1] https://sigsep.github.io/datasets/dsd100.html
[2] http://craffel.github.io/mir_eval
[3] http://librosa.org/doc/main/generated/librosa.stft.html
[4] http://craffel.github.io/mir_eval/#mir_eval.separation.bss_eval_images_framewise
[5] https://sisec18.unmix.app