Advanced Audio Processing: Exercise 04

Audio Classification with data augmentation

1 Introduction

In the previous exercises you have created deep neural networks (DNNs) to do classification and detection. The performance obtained with the previous exercise is not the best one, classification and detection can be improved using different techniques. Some of these techniques are transfer learning, data augmentation and scheduling the learning rate. In this exercise you will focus on techniques related to data augmentation. You will use the previous DNN and implement different data augmentation techniques and evaluate the performance of the system. In all tasks, you are free to use either a CPU or a graphics processing unit (GPU) for implementing the necessary calculations.

The rest of the document is organized as follows. In Section 2 you will set up your data, creating different data augmentation techniques from the audio data. In Section 3 you will try different combinations and analyze the performance.

2 Setting-up the data (0.5 point)

Up to now you have been provided with ready-to-use data. In this exercise, you will be given the raw audio data and you will have to extract the mel-spectrogram features 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.

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 three directories, `training`, `testing` and `validation`. You will have to read the audio files from the directories, extract features, and create the same directory structure that will host your features.
The data used for this exercise is from ESC-50 [1], we have selected a subset from it, specifically four classes: *rain*, *sea waves*, *chainsaw* and *helicopter*. The splits are made so the available training and validation data is reduced.

You will have to:

1. Download and expand the dataset in the root directory of your project for this exercise.
2. Create a dataset class, which will accept an indication if it is training, testing or validation split. Code for this can be found in `dataset_class.py`, as from previous exercises, together with `getting_and_init_the_data.py` for calling the dataset class.
3. Serialize the data after extracting mel band features.

The code implementing the serialization of the data should be placed in a file called **serialize_data_<your name>.py**. A set of functions used in previous exercises, such as mel extraction or getting files from folders are provided in the script utils.py.

3 Data augmentation (1.5 points)

When the model is trained with a very small set of samples per class, it tends to "memorize" the training set rather than generalize, leading to overfitting. A group of most popular augmentation techniques will be covered in this exercise, applying these methods to the set of training data, will result in a bigger dataset.

Specifically, in this part you have to:

1. Implement one function for each of the corresponding augmentation methods
2. Use one or combination of them for training your model

3.1 Basic 1-D alterations(0.5 point)

The most simple methods for data augmentation are adopted from computer vision. Here we will try to implement the most popular ones, such as adding noise and pitch shifting of the signal.

- Adding noise: white noise, brownian noise or even "real" noise extracted from recorded soundscapes labeled as background noises.
- Pitch shift: the audio will sound higher or lower by shifting upwards or downwards the frequency components by a constant factor.

3.2 Reverberation (0.5 points)

A bit more complex technique is to simulate a sound being played in a different environment by using the room impulse response (RIR). In order to get this effect the signal and the RIR from the desired environment are convolved, creating a new synthetically generated signal.

Specifically, you have to:

1. Extract the room impulse response from the **.zip** file, impulse_response.wav. If you wish you can use another RIR from your choice.
2. Implement a function called "add_impulse_response" that convolves the original audio file with the room impulse response.

3.3 specAugment (0.5 points)

The main difference from previous augmentation methods is that this technique is applied to the spectrogram not to the raw signal.

Horizontal flip or different rotations are commonly used in image processing with good results, however the same do not apply to spectrograms, since it will substantially modify the sound that it represents. Instead SpecAugment is used, where different sections of the spectrogram are blocked out. As introduced in [2] specAugment consist of three steps:

1. Time wrapping
2. Frequency mask: where randomly consecutive frequencies are masked out.
3. Time mask: where randomly ranges of time are masked out.

The process of masking out is performed by adding horizontal (frequency mask) or vertical (time mask) bars on the spectrogram, read [2] for more detailed information.

For this exercise we will be implementing Frequency Masking only for simplicity. Specifically, you have to:

1. Randomly select a range of frequency channels that will be masked from the spectrogram.
2. The selected range is then zero-out, mapped to zeroes.

When selecting the range $[f_0, f_0+f]$, f value can be randomly sampled from a uniform distribution from 0 to the frequency max parameter (upper level, e.g. 40). The parameter f_0 is randomly selected within the range [0, mF - f] where mF is the number of mel frequency channels.

The corresponding code and comments should be placed in the file **data_augmentation.py** (please organize your code in functions, with proper/descriptive naming), which you will submit. Any text answers (i.e. not code) should be as comments in the Python file.

4. Train the model (1 point)

The main goal of this exercise is to improve the performance of an audio classification model by using data augmentation techniques. The model used for training will be the one implemented in exercise 2 [3]. The steps are:

1. Implement the training, validation, and testing processes of your system, you can use your own code or the provided code from the previous exercise(s).
2. Set-up the loop over epochs.
3. Set-up the early stopping process.
4. Log training, validation, and testing losses and print them to console (where appropriate).
5. Since the problem is multi-class classification you should use the cross-entropy loss. A part of calculating the loss, for better understanding of the output, calculate the accuracy of the model and represent the confusion matrix [4] (CM) for the testing data. The CM helps to identify which classes are more difficult to classify, giving a more detailed analysis of the model performance.
6. Compare the model performance using data augmentation techniques on the training data against when only the original training samples are used.

Which of the used augmentation techniques has helped to increase the model accuracy? was only one method or did you use a combination of them? comment on this.

You should submit all your code (even the code that you have developed in previous exercises); **training_1_<your name>.py** and **data_augmentation.py** .

References

[1] https://github.com/karolpiczak/ESC-50
[2] https://ai.googleblog.com/2019/04/specaugment-new-data-augmentation.html
[3] https://moodle.tuni.fi/mod/assign/view.php?id=1458043
[4] https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html