**Dataset**

The train and test sets of the dataset are saved in UrbanSound8K\_train.pkl, and UrbanSound8K\_test.pkl.

The dataset is structured as a list of dictionaries. Each dict in the list corresponds to a different audio segment from an audio file. The dicts contain the following keys:

* **filename:** contains a unique name of the audio file. This is useful for matching audio segments to the audio file that they are coming from, and compute global scores by averaging the segments scores that have the same **filename**
* **class:** class name
* **classID:** class number [0…9]
* **features:** all the features to be used for training. This is a dictionary which contains:
  + **logmelspec**
  + **mfcc**
  + **chroma**
  + **spectral\_contrast**
  + **Tonnetz**

**Dataloader**

In dataset.py, the body of a PyTorch dataloader can be found to load UrbanSound8K dataset. **You first have to edit this file** **to load the different inputs (LMC, MC, and MLMC features) for training your convolutional networks. The code already loads the labels, and the unique identifiers of the files that the audio segments belong to. You have to modify the commented lines.**

Then to use it, include the following lines in your code:

from dataset import UrbanSound8KDataset

train\_loader = torch.utils.data.DataLoader(  
 UrbanSound8KDataset(‘UrbanSound8K\_train.pkl’, mode),  
 batch\_size=32, shuffle=**True**,  
 num\_workers=8, pin\_memory=**True**)  
  
val\_loader = torch.utils.data.DataLoader(  
 UrbanSound8KDataset(‘UrbanSound8K\_test.pkl’, mode),  
 batch\_size=32, shuffle=**False**,  
 num\_workers=8, pin\_memory=**True**)

for i, (input, target, filename) **in** enumerate(train\_loader):

*# training code*

for i, (input, target, filename) **in** enumerate(val\_loader):

*# validation code*

In the code above, **input** is a batch of 32 log-mel spectrograms, **target** are their corresponding labels, and **filename** are the names of the audio files each segment belongs to (useful for testing). The variable **mode** should take one of the values: ‘LMC’, ‘MC’, ‘MLMC’.

**Convolutional Neural Network:**

1. Convolutional layer: 1 input channel, 32 output channels, (3x3) kernel size. Size of each channel is 41x85 (input and output).
2. Convolutional layer: 32 input channel, 32 output channels, (3x3) kernel size. Size of each channel is 41x85 (input and output).

* Max pooling: kernel size (2x2) and stride (2x2). Size of input channel is 41x85 and size of output channel is 21x43.

1. Convolutional layer: 32 input channel, 64 output channels, (3x3) kernel size. Size of each channel is 21x43 (input and output).
2. Convolutional layer: 64 input channel, 64 output channels, (3x3) kernel size. Size of each channel is 21x43 (input and output).

* Either max pooling or add stride of step 2 to the fourth layer.

1. Fully Connected layer: 1024 units, with 15488 inputs from the fourth convolutional layer. Sigmoid Activation function.
2. Output layer: 10 units. Softmax activation function.

**Note:** batch-normalization is applied to all convolutional layer.

**Note:** Convolutional layers activation function is ReLu.

**Optimizer:** Adam.

**Loss function:** cross entropy.

**Hyperparameters:** Batch size = 32, learning rate = 0.001, momentum = 0.9.

**Dropout:** 0.5 for second, fourth and fully connected layers.