Freesound Audio Tagging 2019

updated May.02

@fizzbuzz's awesome kernel from previous competition would be a great introduction for beginners, including me :)
(https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data (https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data))
Here I posted the modified kernel for this competition (though not perfect).

Also some top solutions in previous competition will help us.

- 1st solution : https://storage.googleapis.com/kaggle-forum-message-attachments/365414/9991/Jeong_COCAI_task2.pdf (https://storage.googleapis.com/kaggle-forum-message-attachments/365414/9991/Jeong_COCAI_task2.pdf)
- 4th solution : https://www.kaggle.com/c/freesound-audio-tagging/discussion/62634#latest-367166 (https://www.kaggle.com/c/freesound-audio-tagging/discussion/62634#latest-367166)
- 8th solution : https://www.kaggle.com/c/freesound-audio-tagging/discussion/64262#latest-376395 (https://www.kaggle.com/c/freesound-audio-tagging/discussion/64262#latest-376395)
- 11th solution : http://dcase.community/documents/workshop2018/proceedings/DCASE2018Workshop_Wei_100.pdf (http://dcase.community/documents/workshop2018/proceedings/DCASE2018Workshop_Wei_100.pdf)

- DCASE_2018 proceedings : http://dcase.community/workshop2018/proceedings (http://dcase.community/workshop2018/proceedings)

And more...

Planet Understanding the Amazon from Space was a multi-labeled image classification competition.
https://www.kaggle.com/c/planet-understanding-the-amazon-from-space (https://www.kaggle.com/c/planet-understanding-the-amazon-from-space)

1st place solution had been written in Kaggle blog by @bestfitting.
http://blog.kaggle.com/2017/10/17/planet-understanding-the-amazon-from-space-1st-place-winners-interview/ (http://blog.kaggle.com/2017/10/17/planet-understanding-the-amazon-from-space-1st-place-winners-interview/)
Most interesting part for me is the way to consider co-occurence.
In this solution, Ridge regression was used to do it (please read the above material for more detail).

NOTE :
This notebook used only curated wav files,
and did not consider multi-labeled records in train.
For supplement, I have also posted the kernel to explore multi-label audio data. https://www.kaggle.com/maxwell110/explore-multi-labeled-data (https://www.kaggle.com/maxwell110/explore-multi-labeled-data)

Contents

In [1]:
```
# Change this to True for full dataset and learning
COMPLETE_RUN = False
```

Loading data

In [2]:
```
import numpy as np
np.random.seed(1001)

import os
import shutil
import warnings

import IPython
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from tqdm import tqdm_notebook
from sklearn.model_selection import StratifiedKFold

%matplotlib inline
matplotlib.style.use('ggplot')
warnings.filterwarnings("ignore", category=FutureWarning)
```

In [3]:
```
os.listdir('../input/')
```

Out[3]:
```
['train_curated.csv',
 'train_noisy.csv',
 'test',
 'sample_submission.csv',
 'train_curated',
 'train_noisy']
```

In [4]:
```
train = pd.read_csv("../input/train_curated.csv")
test = pd.read_csv("../input/sample_submission.csv")
```

In [5]:
```
train.sample(10)
```

Out[5]:

	fname	labels
1492	4de932d0.wav	Male_speech_and_man_speaking,Female_speech_and...
4727	f396ec97.wav	Traffic_noise_and_roadway_noise
146	07dd3742.wav	Car_passing_by
3217	a6e6c971.wav	Cutlery_and_silverware
4579	ecb1852a.wav	Male_singing
685	24334f89.wav	Toilet_flush
700	25033c9b.wav	Sigh
1286	43788b3b.wav	Stream
1159	3c65185e.wav	Toilet_flush
1963	6564cce8.wav	Harmonica

In [6]:
```
test.sample(5)
```

Out[6]:

	fname	Accelerating_and_revving_and_vroom	Accordion	Acoustic_guitar	Applause	Bark	Ba
134	07dff283.wav	0	0	0	0	0	
866	342cc3ae.wav	0	0	0	0	0	
180	0a64010f.wav	0	0	0	0	0	
185	0aee7a2a.wav	0	0	0	0	0	
901	3623b13a.wav	0	0	0	0	0	

In [7]:
```
print("Number of train examples=", train.shape[0], "  Number of classes=", l
en(set(train.labels)))
print("Number of test examples=", test.shape[0], "  Number of classes=", len
(set(test.columns[1:])))
```

```
Number of train examples= 4970   Number of classes= 213
Number of test examples= 1120   Number of classes= 80
```

Due to multi-labeld records in train, the number of unique classes is 213 (> 80).

Distribution of Categories

For simplicity, we excluded multi-labeled records in train, so the number of unique label is 74 (< 80).
When bulid a valid model, we must consider this.

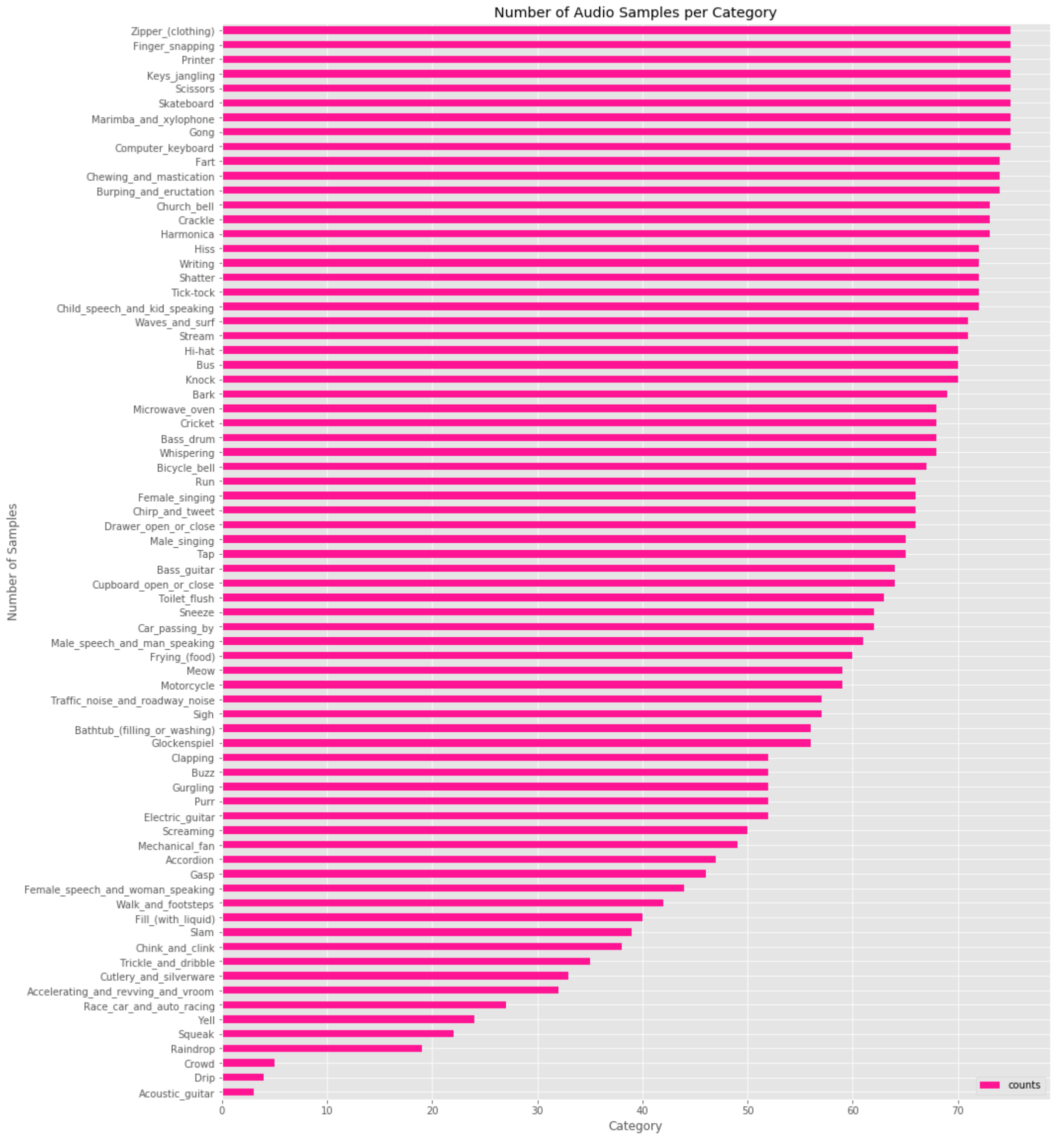
In [8]:
```
train = train[train.labels.isin(test.columns[1:])]
print(len(train))
```

```
4269
```

In [9]:
```
category_group = train.groupby(['labels']).count()
category_group.columns = ['counts']
print(len(category_group))
```

```
74
```

In [10]:
```
plot = category_group.sort_values(ascending=True, by='counts').plot(
    kind='barh',
    title="Number of Audio Samples per Category",
    color='deeppink',
    figsize=(15,20))
plot.set_xlabel("Category")
plot.set_ylabel("Number of Samples");
```

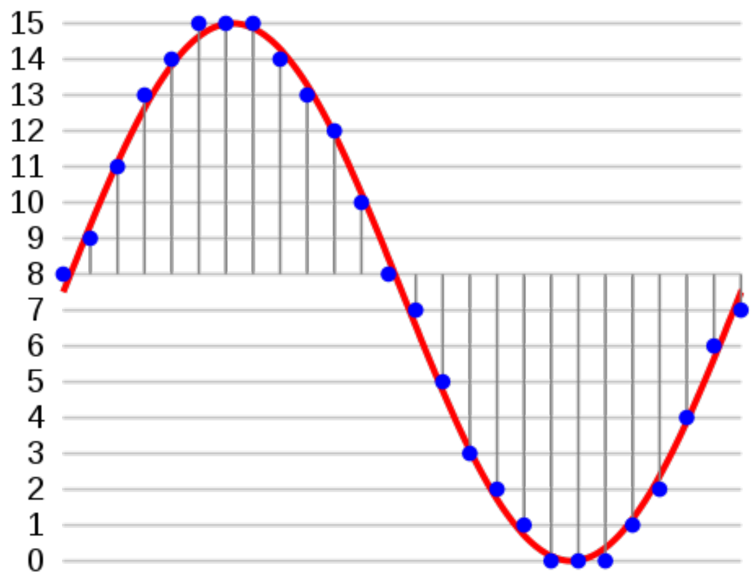


In [11]:
```
print('Minimum samples per category = ', min(train.labels.value_counts()))
print('Maximum samples per category = ', max(train.labels.value_counts()))
```

```
Minimum samples per category =  3
Maximum samples per category =  75
```

Reading Audio Files

The audios are [Pulse-code modulated (https://en.wikipedia.org/wiki/Audio_bit_depth)](https://en.wikipedia.org/wiki/Audio_bit_depth) with a [bit depth (https://en.wikipedia.org/wiki/Audio_bit_depth)](https://en.wikipedia.org/wiki/Audio_bit_depth) of 16 and a [sampling rate (https://en.wikipedia.org/wiki/Sampling_%28signal_processing%29)](https://en.wikipedia.org/wiki/Sampling_%28signal_processing%29) of 44.1 kHz



- **Bit-depth = 16**: The amplitude of each sample in the audio is one of 2^16 (=65536) possible values.
- **Samplig rate = 44.1 kHz**: Each second in the audio consists of 44100 samples. So, if the duration of the audio file is 3.2 seconds, the audio will consist of 44100*3.2 = 141120 values.

Let's listen to an audio file in our dataset and load it to a numpy array

In [12]:
```python
import IPython.display as ipd  # To play sound in the notebook
fname = '../input/train_curated/0019ef41.wav'   # Raindrop
ipd.Audio(fname)
```

Out[12]:

0:00 / 0:02

In [13]:
```python
# Using wave library
import wave
wav = wave.open(fname)
print("Sampling (frame) rate = ", wav.getframerate())
print("Total samples (frames) = ", wav.getnframes())
print("Duration = ", wav.getnframes()/wav.getframerate())
```

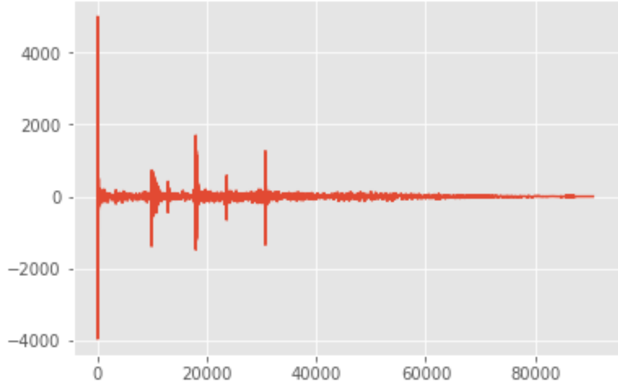
```
Sampling (frame) rate =  44100
Total samples (frames) =  90616
Duration =  2.054784580498866
```

In [14]:
```python
# Using scipy
from scipy.io import wavfile
rate, data = wavfile.read(fname)
print("Sampling (frame) rate = ", rate)
print("Total samples (frames) = ", data.shape)
print(data)
```

```
Sampling (frame) rate =  44100
Total samples (frames) =  (90616,)
[ 369 -577  -49 ...    0    0    0]
```

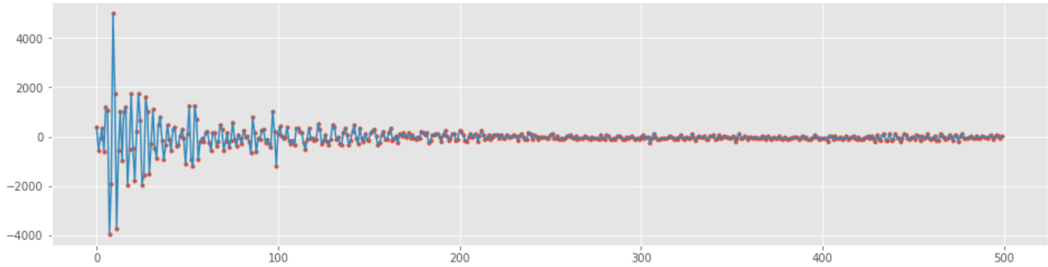
Let's plot the audio frames

In [15]:
```
plt.plot(data, '-', );
```



Let's zoom in on first 1000 frames

In [16]:
```
plt.figure(figsize=(16, 4))
plt.plot(data[:500], '.'); plt.plot(data[:500], '-');
```



Audio Length

We shall now analyze the lengths of the audio files in our dataset

In [17]:
```
train['nframes'] = train['fname'].apply(lambda f: wave.open('../input/train_
curated/' + f).getnframes())
test['nframes'] = test['fname'].apply(lambda f: wave.open('../input/test/' +
f).getnframes())
```

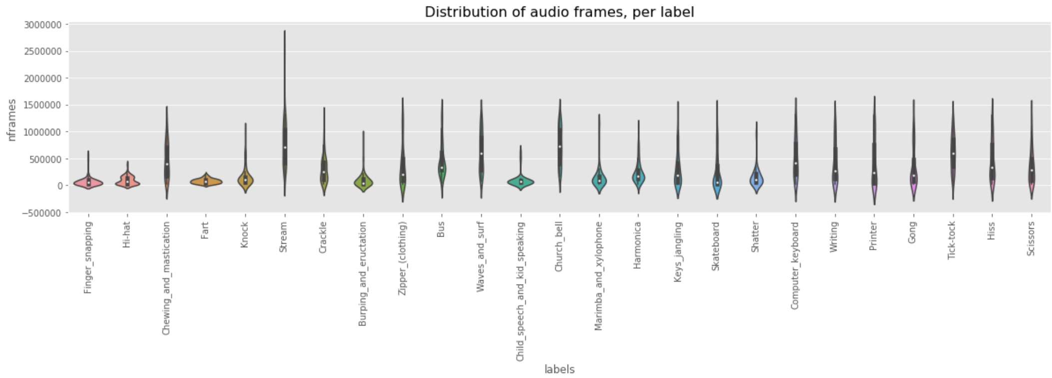
In [18]:
```
train.head()
```

Out[18]:

	fname	labels	nframes
0	0006ae4e.wav	Bark	310456
1	0019ef41.wav	Raindrop	90616
2	001ec0ad.wav	Finger_snapping	66976
3	0026c7cb.wav	Run	1125886
4	0026f116.wav	Finger_snapping	60638

The number of categories is large, so let's check the frame distributions of top 25 categories.

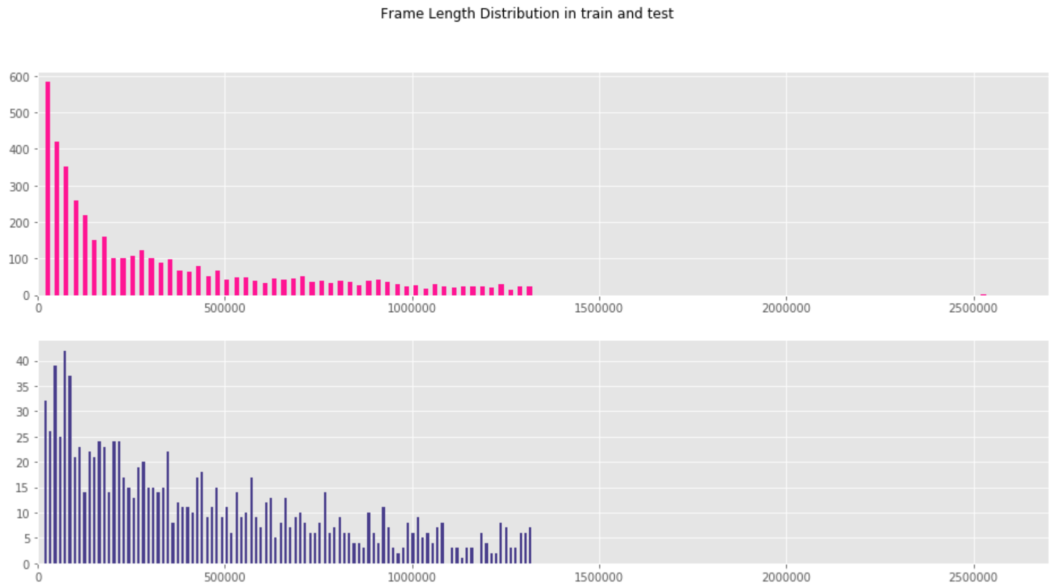
In [19]:
```
idx_sel = category_group.sort_values(ascending=True, by='counts').index[-2
5:]
_, ax = plt.subplots(figsize=(20, 4))
sns.violinplot(ax=ax, x="labels", y="nframes", data=train[(train.labels.isin
(idx_sel).values)])
plt.xticks(rotation=90)
plt.title('Distribution of audio frames, per label', fontsize=16)
plt.show()
```



We observe:
The distribution of audio length across labels is non-uniform and has high variance as the previous competition.

Let's now analyze the frame length distribution in train and test.

In [20]:
```
fig, ax = plt.subplots(2, 1, figsize=(16,8))
train.nframes.hist(bins=100, grid=True, rwidth=0.5, ax=ax[0], color='deeppin
k')
test.nframes.hist(bins=100, grid=True, rwidth=0.5, ax=ax[1], color='darkslat
eblue')
ax[0].set_xlim(0, 2700000)
ax[1].set_xlim(0, 2700000)
plt.suptitle('Frame Length Distribution in train and test', ha='center', fon
tsize='large');
```



We observe:

- Majority of the audio files are short.
- There are an `abnormal` length in the train histogram. Let's analyze them.

In [21]:
```
train.query("nframes > 2500000")
```

Out[21]:

	fname	labels	nframes
2316	77b925c2.wav	Stream	2538889

In [22]:
```
abnormal_length = 2538889

abnormal_fnames = train.loc[train.nframes == abnormal_length, 'fname'].value
s[0]
ipd.Audio( '../input/train_curated/' + abnormal_fnames)
```

Out[22]:

0:00 / 0:58

2. Building a Model using Raw Wave

We will build two models:

1. The first model will take the raw audio (1D array) as input and the primary operation will be Conv1D
2. The second model will take the MFCCs as input. (We will explain MFCC later)

Keras Model using raw wave

Our model has the architecture as follows:

rav

Important: Due to the time limit on Kaggle Kernels, it is not possible to perform 10-fold training of a large model. I have trained the model locally and uploaded its output files as a dataset. If you wish to train the bigger model, change `COMPLETE_RUN = True` at the beginning of the kernel.

Some sssential imports

In [23]:
```
import librosa
import numpy as np
import scipy
from keras import losses, models, optimizers
from keras.activations import relu, softmax
from keras.callbacks import (EarlyStopping, LearningRateScheduler,
                             ModelCheckpoint, TensorBoard, ReduceLROnPlatea
u)
from keras.layers import (Convolution1D, Dense, Dropout, GlobalAveragePoolin
g1D,
                          GlobalMaxPool1D, Input, MaxPool1D, concatenate)
from keras.utils import Sequence, to_categorical
```

```
Using TensorFlow backend.
```

Configuration

The Configuration object stores those learning parameters that are shared between data generators, models, and training functions. Anything that is `global` as far as the training is concerned can become the part of Configuration object.

In [24]:
```
class Config(object):
    def __init__(self,
                 sampling_rate=16000, audio_duration=2,
                 n_classes=len(category_group),
                 use_mfcc=False, n_folds=10, learning_rate=0.0001,
                 max_epochs=50, n_mfcc=20):
        self.sampling_rate = sampling_rate
        self.audio_duration = audio_duration
        self.n_classes = n_classes
        self.use_mfcc = use_mfcc
        self.n_mfcc = n_mfcc
        self.n_folds = n_folds
        self.learning_rate = learning_rate
        self.max_epochs = max_epochs

        self.audio_length = self.sampling_rate * self.audio_duration
        if self.use_mfcc:
            self.dim = (self.n_mfcc, 1 + int(np.floor(self.audio_length/51
2)), 1)
        else:
            self.dim = (self.audio_length, 1)
```

DataGenerator Class

The DataGenerator class inherits from **`keras.utils.Sequence`** . It is useful for preprocessing and feeding the data to a Keras model.

- Once initialized with a batch_size, it computes the number of batches in an epoch. The **`__len__`** method tells Keras how many batches to draw in each epoch.
- The **`__getitem__`** method takes an index (which is the batch number) and returns a batch of the data (both X and y) after calculating the offset. During test time, only `X` is returned.
- If we want to perform some action after each epoch (like shuffle the data, or increase the proportion of augmented data), we can use the **`on_epoch_end`** method.

Note: **`Sequence`** are a safer way to do multiprocessing. This structure guarantees that the network will only train once on each sample per epoch which is not the case with generators.

In [25]:
```
class DataGenerator(Sequence):
    def __init__(self, config, data_dir, list_IDs, labels=None,
                 batch_size=64, preprocessing_fn=lambda x: x):
        self.config = config
        self.data_dir = data_dir
        self.list_IDs = list_IDs
        self.labels = labels
        self.batch_size = batch_size
        self.preprocessing_fn = preprocessing_fn
        self.on_epoch_end()
        self.dim = self.config.dim

    def __len__(self):
        return int(np.ceil(len(self.list_IDs) / self.batch_size))

    def __getitem__(self, index):
        indexes = self.indexes[index*self.batch_size:(index+1)*self.batch_si
ze]
        list_IDs_temp = [self.list_IDs[k] for k in indexes]
        return self.__data_generation(list_IDs_temp)

    def on_epoch_end(self):
        self.indexes = np.arange(len(self.list_IDs))

    def __data_generation(self, list_IDs_temp):
        cur_batch_size = len(list_IDs_temp)
        X = np.empty((cur_batch_size, *self.dim))

        input_length = self.config.audio_length
        for i, ID in enumerate(list_IDs_temp):
            file_path = self.data_dir + ID

            # Read and Resample the audio
            data, _ = librosa.core.load(file_path, sr=self.config.sampling_r
ate,
                                        res_type='kaiser_fast')

            # Random offset / Padding
            if len(data) > input_length:
                max_offset = len(data) - input_length
                offset = np.random.randint(max_offset)
                data = data[offset:(input_length+offset)]
            else:
                if input_length > len(data):
                    max_offset = input_length - len(data)
                    offset = np.random.randint(max_offset)
                else:
                    offset = 0
                data = np.pad(data, (offset, input_length - len(data) - offs
et), "constant")

            # Normalization + Other Preprocessing
            if self.config.use_mfcc:
                data = librosa.feature.mfcc(data, sr=self.config.sampling_ra
te,
                                                   n_mfcc=self.config.n_mfc
c)
                data = np.expand_dims(data, axis=-1)
            else:
                data = self.preprocessing_fn(data)[:, np.newaxis]
            X[i,] = data

        if self.labels is not None:
            y = np.empty(cur_batch_size, dtype=int)
            for i, ID in enumerate(list_IDs_temp):
                y[i] = self.labels[ID]
            return X, to_categorical(y, num_classes=self.config.n_classes)
        else:
```

Normalization

Normalization is a crucial preprocessing step. The simplest method is rescaling the range of features to scale the range in [0, 1].

In [26]:
```
def audio_norm(data):
    max_data = np.max(data)
    min_data = np.min(data)
    data = (data-min_data)/(max_data-min_data+1e-6)
    return data - 0.5
```

- The dummy model is just for debugging purpose.
- Our 1D Conv model is fairly deep and is trained using Adam Optimizer with a learning rate of 0.0001

In [27]:
```
def get_1d_dummy_model(config):

    nclass = config.n_classes
    input_length = config.audio_length

    inp = Input(shape=(input_length,1))
    x = GlobalMaxPool1D()(inp)
    out = Dense(nclass, activation=softmax)(x)

    model = models.Model(inputs=inp, outputs=out)
    opt = optimizers.Adam(config.learning_rate)

    model.compile(optimizer=opt, loss=losses.categorical_crossentropy, metri
cs=['acc'])
    return model

def get_1d_conv_model(config):

    nclass = config.n_classes
    input_length = config.audio_length

    inp = Input(shape=(input_length,1))
    x = Convolution1D(16, 9, activation=relu, padding="valid")(inp)
    x = Convolution1D(16, 9, activation=relu, padding="valid")(x)
    x = MaxPool1D(16)(x)
    x = Dropout(rate=0.1)(x)

    x = Convolution1D(32, 3, activation=relu, padding="valid")(x)
    x = Convolution1D(32, 3, activation=relu, padding="valid")(x)
    x = MaxPool1D(4)(x)
    x = Dropout(rate=0.1)(x)

    x = Convolution1D(32, 3, activation=relu, padding="valid")(x)
    x = Convolution1D(32, 3, activation=relu, padding="valid")(x)
    x = MaxPool1D(4)(x)
    x = Dropout(rate=0.1)(x)

    x = Convolution1D(256, 3, activation=relu, padding="valid")(x)
    x = Convolution1D(256, 3, activation=relu, padding="valid")(x)
    x = GlobalMaxPool1D()(x)
    x = Dropout(rate=0.2)(x)

    x = Dense(64, activation=relu)(x)
    x = Dense(1028, activation=relu)(x)
    out = Dense(nclass, activation=softmax)(x)

    model = models.Model(inputs=inp, outputs=out)
    opt = optimizers.Adam(config.learning_rate)

    model.compile(optimizer=opt, loss=losses.categorical_crossentropy, metri
cs=['acc'])
    return model
```

Training 1D Conv

It is important to convert raw labels to integer indices

In [28]:
```
train.head()
```

Out[28]:

	fname	labels	nframes
0	0006ae4e.wav	Bark	310456
1	0019ef41.wav	Raindrop	90616
2	001ec0ad.wav	Finger_snapping	66976
3	0026c7cb.wav	Run	1125886
4	0026f116.wav	Finger_snapping	60638

In [29]:
```
LABELS = list(train.labels.unique())
label_idx = {label: i for i, label in enumerate(LABELS)}
train.set_index("fname", inplace=True)
test.set_index("fname", inplace=True)
train["label_idx"] = train.labels.apply(lambda x: label_idx[x])
if not COMPLETE_RUN:
    train = train[:2000]
    test = test[:2000]
```

In [30]:
```
train.head()
```

Out[30]:

	labels	nframes	label_idx
fname			
0006ae4e.wav	Bark	310456	0
0019ef41.wav	Raindrop	90616	1
001ec0ad.wav	Finger_snapping	66976	2
0026c7cb.wav	Run	1125886	3
0026f116.wav	Finger_snapping	60638	2

In [31]:
```
config = Config(sampling_rate=16000, audio_duration=2, n_folds=10, learning_
rate=0.001)
if not COMPLETE_RUN:
    config = Config(sampling_rate=100, audio_duration=1, n_folds=2, max_epoc
hs=1)
```

Here is the code for 10-fold training:

- We use **`from sklearn.model_selection.StratifiedKFold`** for splitting the trainig data into 10 folds.
- We use some Keras callbacks to monitor the training.
  - **`ModelCheckpoint`** saves the best weight of our model (using validation data). We use this weight to make test predictions.
  - **`EarlyStopping`** stops the training once validation loss ceases to decrease
  - **`TensorBoard`** helps us visualize training and validation loss and accuracy.
- We fit the model using **`DataGenerator`** for training and validation splits.
- We get both training and test predictions and save them as .npy format. We also generate a submission file. For 10-fold CV, the number of prediction files should be 10. We will ensemble these predictions later.

In [32]:
```
PREDICTION_FOLDER = "predictions_1d_conv"
if not os.path.exists(PREDICTION_FOLDER):
    os.mkdir(PREDICTION_FOLDER)
if os.path.exists('logs/' + PREDICTION_FOLDER):
    shutil.rmtree('logs/' + PREDICTION_FOLDER)

skf = StratifiedKFold(n_splits=config.n_folds)

for i, (train_split, val_split) in enumerate(skf.split(train.index, train.la
bel_idx)):
    train_set = train.iloc[train_split]
    val_set = train.iloc[val_split]
    checkpoint = ModelCheckpoint('best_%d.h5'%i, monitor='val_loss', verbos
e=1, save_best_only=True)
    early = EarlyStopping(monitor="val_loss", mode="min", patience=5)
    tb = TensorBoard(log_dir='./logs/' + PREDICTION_FOLDER + '/fold_%d'%i, w
rite_graph=True)

    callbacks_list = [checkpoint, early, tb]
    print("\nFold: ", i)
    if COMPLETE_RUN:
        model = get_1d_conv_model(config)
    else:
        model = get_1d_dummy_model(config)

    train_generator = DataGenerator(config, '../input/train_curated/', train
_set.index,
                                    train_set.label_idx, batch_size=64,
                                    preprocessing_fn=audio_norm)
    val_generator = DataGenerator(config, '../input/train_curated/', val_se
t.index,
                                  val_set.label_idx, batch_size=64,
                                  preprocessing_fn=audio_norm)

    history = model.fit_generator(train_generator, callbacks=callbacks_list,
validation_data=val_generator,
                                  epochs=config.max_epochs, use_multiprocess
ing=True, max_queue_size=20)

#     model.load_weights('../working/best_%d.h5'%i)

    # Save train predictions
    train_generator = DataGenerator(config, '../input/train_curated/', trai
n.index, batch_size=128,
                                    preprocessing_fn=audio_norm)
    predictions = model.predict_generator(train_generator, use_multiprocessi
ng=True,
                                          max_queue_size=20, verbose=1)
    np.save(PREDICTION_FOLDER + "/train_predictions_%d.npy"%i, predictions)

    # Save test predictions
    test_generator = DataGenerator(config, '../input/test/', test.index, bat
ch_size=128,
                                    preprocessing_fn=audio_norm)
    predictions = model.predict_generator(test_generator, use_multiprocessin
g=True,
                                          max_queue_size=20, verbose=1)
    np.save(PREDICTION_FOLDER + "/test_predictions_%d.npy"%i, predictions)

    # Make a submission file
    top_3 = np.array(LABELS)[np.argsort(-predictions, axis=1)[:, :3]]
    predicted_labels = [' '.join(list(x)) for x in top_3]
    test['label'] = predicted_labels
    test[['label']].to_csv(PREDICTION_FOLDER + "/predictions_%d.csv"%i)
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/model_selection/_split.py:652:
Warning: The least populated class in y has only 1 members, which is too few.
The minimum number of members in any class cannot be less than n_splits=2.
  % (min_groups, self.n_splits)), Warning)

Fold:  0
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/pyt
hon/framework/op_def_library.py:263: colocate_with (from tensorflow.python.fr
amework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/pyt
hon/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is d
eprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/1
16/16 [==============================] - 120s 8s/step - loss: 4.3050 - acc:
0.0127 - val_loss: 4.3058 - val_acc: 0.0128

Epoch 00001: val_loss improved from inf to 4.30585, saving model to best_0.h5
16/16 [==============================] - 233s 15s/step
9/9 [==============================] - 100s 11s/step

Fold:  1
Epoch 1/1
16/16 [==============================] - 109s 7s/step - loss: 4.3089 - acc:
0.0127 - val_loss: 4.3086 - val_acc: 0.0133

Epoch 00001: val_loss improved from inf to 4.30861, saving model to best_1.h5
Epoch 1/1
16/16 [==============================] - 225s 14s/step
9/9 [==============================] - 92s 10s/step
```

predictions are saved as following.

In [33]:
```
os.listdir('../working/predictions_1d_conv/')
```

Out[33]:
```
['predictions_0.csv',
 'train_predictions_1.npy',
 'predictions_1.csv',
 'train_predictions_0.npy',
 'test_predictions_0.npy',
 'test_predictions_1.npy']
```

Ensembling 1D Conv Predictions

Now that we have trained our model, it is time average the predictions of X-folds.
We will try **Geometric Mean averaging**.

In [34]:
```
pred_list = []
for i in range(config.n_folds):
    pred_list.append(np.load("../working/predictions_1d_conv/test_prediction
s_%d.npy"%i))
prediction = np.ones_like(pred_list[0])
for pred in pred_list:
    prediction = prediction*pred
prediction = prediction**(1./len(pred_list))
# Make a submission file
top_3 = np.array(LABELS)[np.argsort(-prediction, axis=1)[:, :3]]
predicted_labels = [' '.join(list(x)) for x in top_3]
test = pd.read_csv('../input/sample_submission.csv')
test['label'] = predicted_labels
test[['fname', 'label']].to_csv("1d_conv_ensembled_submission.csv", index=Fa
lse)
```

3. Introuction to MFCC

As we have seen in the previous section, our Deep Learning models are powerful enough to classify sounds from the raw audio. We do not require any complex feature engineering. But before the Deep Learning era, people developed techniques to extract features from audio signals. It turns out that these techniques are still useful. One such technique is computing the MFCC (Mel Frquency Cepstral Coefficients) from the raw audio. Before we jump to MFCC, let's talk about extracting features from the sound.

If we just want to classify some sound, we should build features that are **speaker independent**. Any feature that only gives information about the speaker (like the pitch of their voice) will not be helpful for classification. In other words, we should extract features that depend on the "content" of the audio rather than the nature of the speaker. Also, a good feature extraction technique should mimic the human speech perception. We don't hear loudness on a linear scale. If we want to double the perceived loudness of a sound, we have to put 8 times as much energy into it. Instead of a linear scale, our perception system uses a log scale.

Taking these things into account, Davis and Mermelstein came up with MFCC in the 1980's. MFCC mimics the logarithmic perception of loudness and pitch of human auditory system and tries to eliminate speaker dependent characteristics by excluding the fundamental frequency and their harmonics. The underlying mathematics is quite complicated and we will skip that. For those interested, here is the [detailed explanation (http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/)](http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/).

Generating MFCC using Librosa

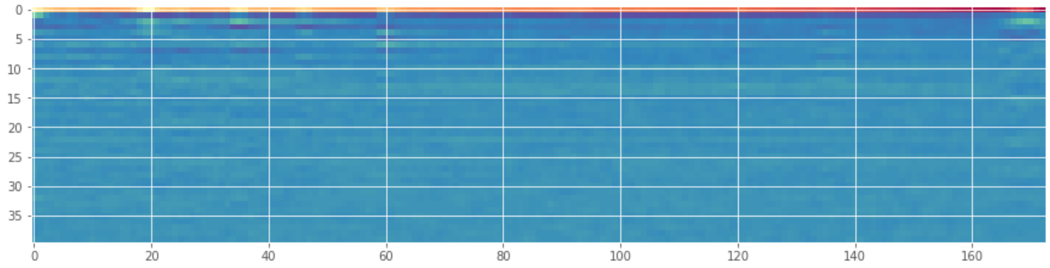
The library librosa has a function to calculate MFCC. Let's compute the MFCC of an audio file and visualize it.

In [35]:
```
import librosa
SAMPLE_RATE = 44100
fname = '../input/train_curated/0019ef41.wav'  # Raindrop
wav, _ = librosa.core.load(fname, sr=SAMPLE_RATE)
wav = wav[:2*44100]
```

In [36]:
```
mfcc = librosa.feature.mfcc(wav, sr = SAMPLE_RATE, n_mfcc=40)
mfcc.shape
```

Out[36]:
```
(40, 173)
```

In [37]:
```
fig, ax = plt.subplots(figsize=(15, 5))
ax.imshow(mfcc, cmap='Spectral', interpolation='nearest');
```



4. Building a Model using MFCC

We will build a 2D Convolutional model using MFCC.

In [38]:
```
from keras.layers import (Convolution2D, GlobalAveragePooling2D, BatchNormal
ization, Flatten,
                          GlobalMaxPool2D, MaxPool2D, concatenate, Activatio
n)
from keras.utils import Sequence, to_categorical
from keras import backend as K
```

In [39]:
```
def get_2d_dummy_model(config):

    nclass = config.n_classes

    inp = Input(shape=(config.dim[0],config.dim[1],1))
    x = GlobalMaxPool2D()(inp)
    out = Dense(nclass, activation=softmax)(x)

    model = models.Model(inputs=inp, outputs=out)
    opt = optimizers.Adam(config.learning_rate)

    model.compile(optimizer=opt, loss=losses.categorical_crossentropy, metri
cs=['acc'])
    return model


def get_2d_conv_model(config):

    nclass = config.n_classes

    inp = Input(shape=(config.dim[0],config.dim[1],1))
    x = Convolution2D(32, (4,10), padding="same")(inp)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = MaxPool2D()(x)

    x = Convolution2D(32, (4,10), padding="same")(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = MaxPool2D()(x)

    x = Convolution2D(32, (4,10), padding="same")(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = MaxPool2D()(x)

    x = Convolution2D(32, (4,10), padding="same")(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = MaxPool2D()(x)

    x = Flatten()(x)
    x = Dense(64)(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    out = Dense(nclass, activation=softmax)(x)

    model = models.Model(inputs=inp, outputs=out)
    opt = optimizers.Adam(config.learning_rate)

    model.compile(optimizer=opt, loss=losses.categorical_crossentropy, metri
cs=['acc'])
    return model
```

Preparing data

In [40]:
```
config = Config(sampling_rate=44100, audio_duration=2, n_folds=10,
                learning_rate=0.001, use_mfcc=True, n_mfcc=40)
if not COMPLETE_RUN:
    config = Config(sampling_rate=44100, audio_duration=2, n_folds=2,
                    max_epochs=1, use_mfcc=True, n_mfcc=40)
```

In [41]:
```
def prepare_data(df, config, data_dir):
    X = np.empty(shape=(df.shape[0], config.dim[0], config.dim[1], 1))
    input_length = config.audio_length
    for i, fname in enumerate(df.index):
#         print(fname)
        file_path = data_dir + fname
        data, _ = librosa.core.load(file_path, sr=config.sampling_rate, res_
type="kaiser_fast")

        # Random offset / Padding
        if len(data) > input_length:
            max_offset = len(data) - input_length
            offset = np.random.randint(max_offset)
            data = data[offset:(input_length+offset)]
        else:
            if input_length > len(data):
                max_offset = input_length - len(data)
                offset = np.random.randint(max_offset)
            else:
                offset = 0
            data = np.pad(data, (offset, input_length - len(data) - offset),
"constant")

        data = librosa.feature.mfcc(data, sr=config.sampling_rate, n_mfcc=co
nfig.n_mfcc)
        data = np.expand_dims(data, axis=-1)
        X[i,] = data
    return X
```

In [42]:
```
test.index = test.fname
```

In [43]:
```
%time
X_train = prepare_data(train, config, '../input/train_curated/')
X_test = prepare_data(test, config, '../input/test/')
y_train = to_categorical(train.label_idx.astype('str'), num_classes=config.n
_classes)
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 23.1 µs
```

Normalization

In [44]:
```
mean = np.mean(X_train, axis=0)
std = np.std(X_train, axis=0)

X_train = (X_train - mean)/std
X_test = (X_test - mean)/std
```

Training 2D Conv on MFCC

In [45]:
```
PREDICTION_FOLDER = "predictions_2d_conv"
if not os.path.exists(PREDICTION_FOLDER):
    os.mkdir(PREDICTION_FOLDER)
if os.path.exists('logs/' + PREDICTION_FOLDER):
    shutil.rmtree('logs/' + PREDICTION_FOLDER)

skf = StratifiedKFold(n_splits=config.n_folds)

for i, (train_split, val_split) in enumerate(skf.split(train.index, train.la
bel_idx)):
    K.clear_session()
    X, y, X_val, y_val = X_train[train_split], y_train[train_split], X_train
[val_split], y_train[val_split]
    checkpoint = ModelCheckpoint('best_%d.h5'%i, monitor='val_loss', verbos
e=1, save_best_only=True)
    early = EarlyStopping(monitor="val_loss", mode="min", patience=5)
    tb = TensorBoard(log_dir='./logs/' + PREDICTION_FOLDER + '/fold_%i'%i, w
rite_graph=True)
    callbacks_list = [checkpoint, early, tb]
    print("#"*50)
    print("Fold: ", i)
    model = get_2d_conv_model(config)
    history = model.fit(X, y, validation_data=(X_val, y_val), callbacks=call
backs_list,
                        batch_size=64, epochs=config.max_epochs)
    model.load_weights('best_%d.h5'%i)

    # Save train predictions
    predictions = model.predict(X_train, batch_size=64, verbose=1)
    np.save(PREDICTION_FOLDER + "/train_predictions_%d.npy"%i, predictions)

    # Save test predictions
    predictions = model.predict(X_test, batch_size=64, verbose=1)
    np.save(PREDICTION_FOLDER + "/test_predictions_%d.npy"%i, predictions)

    # Make a submission file
    top_3 = np.array(LABELS)[np.argsort(-predictions, axis=1)[:, :3]]
    predicted_labels = [' '.join(list(x)) for x in top_3]
    test['label'] = predicted_labels
    test[['label']].to_csv(PREDICTION_FOLDER + "/predictions_%d.csv"%i)
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/model_selection/_split.py:652:
Warning: The least populated class in y has only 1 members, which is too few.
The minimum number of members in any class cannot be less than n_splits=2.
  % (min_groups, self.n_splits)), Warning)

##################################################
Fold:  0
Train on 981 samples, validate on 1019 samples
Epoch 1/1
981/981 [==============================] - 3s 4ms/step - loss: 4.4982 - acc:
0.0183 - val_loss: 4.5130 - val_acc: 0.0343

Epoch 00001: val_loss improved from inf to 4.51296, saving model to best_0.h5
2000/2000 [==============================] - 0s 160us/step
1120/1120 [==============================] - 0s 111us/step
##################################################
Fold:  1
Train on 1019 samples, validate on 981 samples
Epoch 1/1
1019/1019 [==============================] - 1s 1ms/step - loss: 4.4766 - ac
c: 0.0186 - val_loss: 4.4547 - val_acc: 0.0204

Epoch 00001: val_loss improved from inf to 4.45468, saving model to best_1.h5
2000/2000 [==============================] - 0s 155us/step
1120/1120 [==============================] - 0s 103us/step
```

Ensembling 2D Conv Predictions

In [46]:
```
pred_list = []
for i in range(config.n_folds):
    pred_list.append(np.load("../working/predictions_2d_conv/test_prediction
s_%d.npy"%i))
prediction = np.ones_like(pred_list[0])
for pred in pred_list:
    prediction = prediction*pred
prediction = prediction**(1./len(pred_list))
# Make a submission file
top_3 = np.array(LABELS)[np.argsort(-prediction, axis=1)[:, :3]]
predicted_labels = [' '.join(list(x)) for x in top_3]
test = pd.read_csv('../input/sample_submission.csv')
test['label'] = predicted_labels
test[['fname', 'label']].to_csv("2d_conv_ensembled_submission.csv", index=Fa
lse)
```

5. Ensembling 1D Conv and 2D Conv Predictions

In [47]:
```
pred_list = []
for i in range(config.n_folds):
    pred_list.append(np.load("../working/predictions_1d_conv/test_prediction
s_%d.npy"%i))
for i in range(config.n_folds):
    pred_list.append(np.load("../working/predictions_2d_conv/test_prediction
s_%d.npy"%i))
prediction = np.ones_like(pred_list[0])
for pred in pred_list:
    prediction = prediction*pred
prediction = prediction**(1./len(pred_list))
```

Be careful :
Because we exclude multi-labeled records, prediction shape became invalid.

In [48]:
```
prediction.shape
```

Out[48]:
```
(1120, 74)
```