# Beginner's Guide to Audio Data

<https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data>

# Freesound General-Purpose Audio Tagging Challenge

Freesound is a collaborative database of Creative Commons Licensed sounds. The aim of this competition is to classify audio files that cover real-world sounds from musical instruments, humans, animals, machines, etc. Few of the labels are: Trumpet, Squeak, Meow, Applause and Finger\_sapping. One of the challenges is that not all labels are manually verified. A creative solution should be able to partially rely on these weak annotations.

Let's take a tour of the data visualization and model building through this kernel. If you like this work, please show your support by upvotes. Happy Kaggling!

### Contents

1. [Exploratory Data Analysis](https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data" \l "eda)
   * [Loading data](https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data" \l "loading_data)
   * [Distribution of Categories](https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data" \l "distribution)
   * [Reading Audio Files](https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data" \l "audio_files)
   * [Audio Length](https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data" \l "audio_length)
2. [Building a Model using Raw Wave](https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data" \l "1d_model_building)
   * [Model Discription](https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data" \l "1d_discription)
   * [Configuration](https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data" \l "configuration)
   * [DataGenerator class](https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data" \l "data_generator)
   * [Normalization](https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data" \l "1d_normalization)
   * [Training 1D Conv](https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data" \l "1d_training)
   * [Ensembling 1D Conv Predictions](https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data" \l "1d_ensembling)
3. [Introduction to MFCC](https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data" \l "intro_mfcc)
   * [Generating MFCC using Librosa](https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data" \l "librosa_mfcc)
4. [Building a Model using MFCC](https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data" \l "2d_model_building)
   * [Preparing Data](https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data" \l "2d_data)
   * [Normalization](https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data" \l "2d_normalization)
   * [Training 2D Conv on MFCC](https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data" \l "2d_training)
   * [Ensembling 2D Conv Predictions](https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data" \l "2d_ensembling)
5. [Ensembling 1D Conv and 2D Conv Predictions](https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data" \l "1d_2d_ensembling)
6. [Results and Conclusion](https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data" \l "conclusion)

## 1. Exploratory Data Analysis

In [1]:

# Change this to True to replicate the result

COMPLETE\_RUN = False

### Loading data

Output

In [2]:

import numpy as np

np.random.seed(1001)

import os

import shutil

import IPython

import matplotlib

import matplotlib.pyplot as plt

import pandas as pd

import seaborn as sns

from tqdm import tqdm\_notebook

from sklearn.cross\_validation import StratifiedKFold

%matplotlib inline

matplotlib.style.use('ggplot')

Output

In [3]:

train = pd.read\_csv("../input/freesound-audio-tagging/train.csv")

test = pd.read\_csv("../input/freesound-audio-tagging/sample\_submission.csv")

Output

In [4]:

train.head()

Output

In [5]:

print("Number of training examples=", train.shape[0], " Number of classes=", len(train.label.unique()))

Output

In [6]:

print(train.label.unique())

### Distribution of Categories

In [7]:

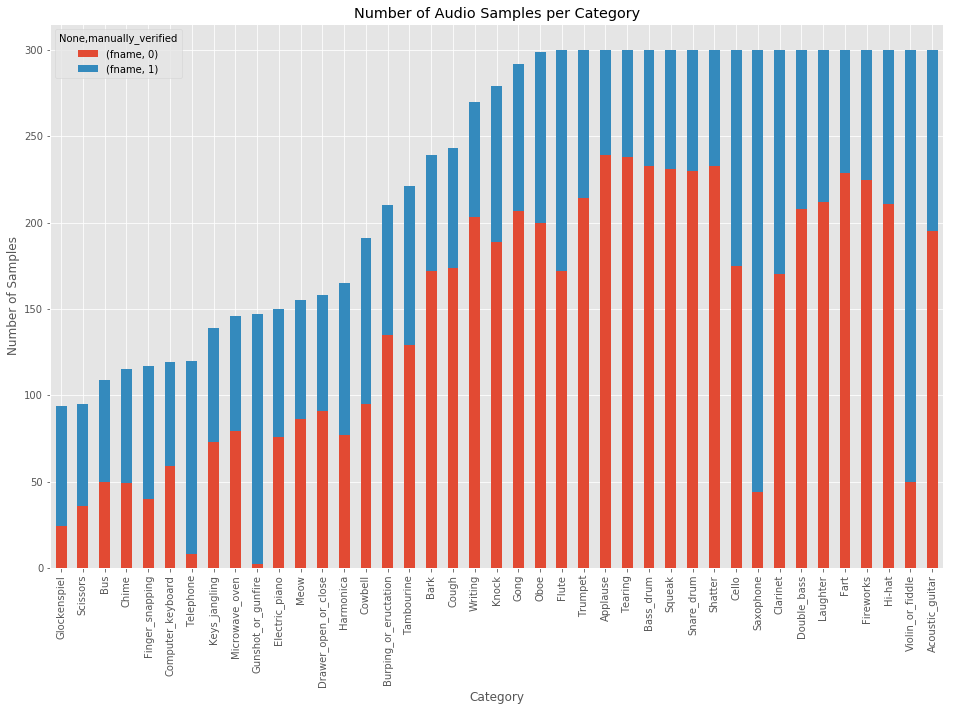
category\_group = train.groupby(['label', 'manually\_verified']).count()

plot = category\_group.unstack().reindex(category\_group.unstack().sum(axis=1).sort\_values().index)\

.plot(kind='bar', stacked=True, title="Number of Audio Samples per Category", figsize=(16,10))

plot.set\_xlabel("Category")

plot.set\_ylabel("Number of Samples");



Output

In [8]:

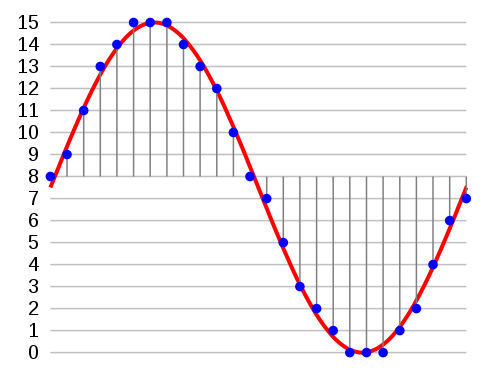
print('Minimum samples per category = ', min(train.label.value\_counts()))

print('Maximum samples per category = ', max(train.label.value\_counts()))

We observe that:

1. The number of audio samples per category is non-nform. The minimum number of audio samples in a category is 94 while the maximum is 300
2. Also, the proportion of maually\_verified labels per category is non-uniform. ### Reading Audio Files

The audios are [Pulse-code modulated](https://en.wikipedia.org/wiki/Audio_bit_depth) with a [bit depth](https://en.wikipedia.org/wiki/Audio_bit_depth) of 16 and a [sampling rate](https://en.wikipedia.org/wiki/Sampling_(signal_processing)) 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 [9]:

import IPython.display as ipd # To play sound in the notebook

fname = '../input/freesound-audio-tagging/audio\_train/' + '00044347.wav' # Hi-hat

ipd.Audio(fname)

Out[9]:

In [10]:

# 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) = 617400

Duration = 14.0

In [11]:

# 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

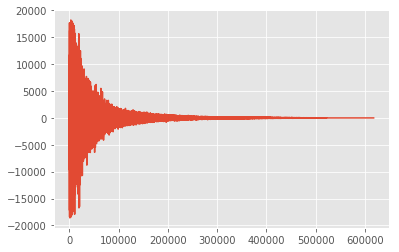
Total samples (frames) = (617400,)

[ 0 26 -5 ... 1 0 0]

Let's plot the audio frames

In [12]:

plt.plot(data, '-', );

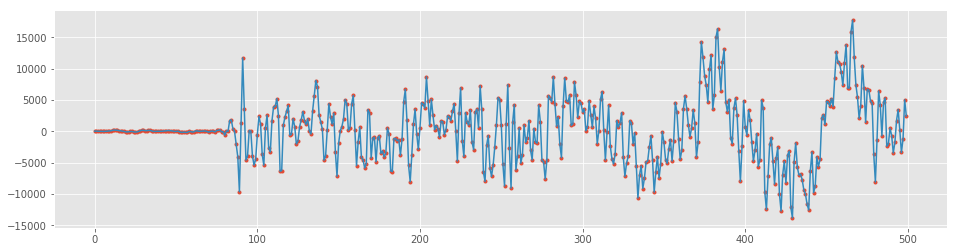


Let's zoom in on first 1000 frames

In [13]:

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 [14]:

train['nframes'] = train['fname'].apply(lambda f: wave.open('../input/freesound-audio-tagging/audio\_train/' + f).getnframes())

test['nframes'] = test['fname'].apply(lambda f: wave.open('../input/freesound-audio-tagging/audio\_test/' + f).getnframes())

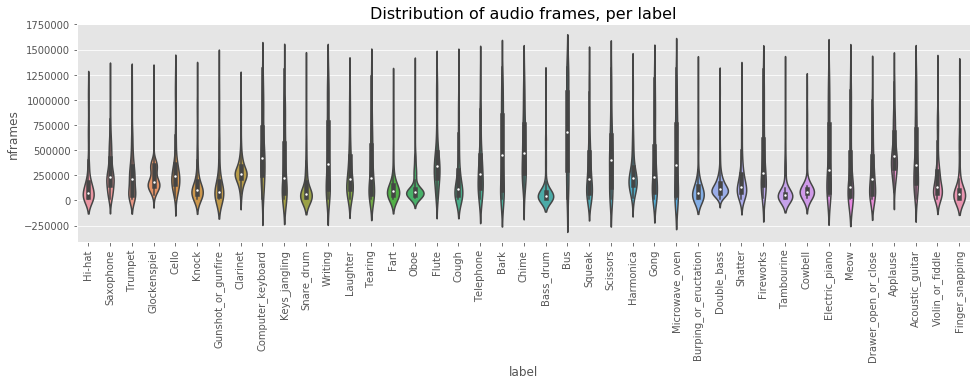
\_, ax = plt.subplots(figsize=(16, 4))

sns.violinplot(ax=ax, x="label", y="nframes", data=train)

plt.xticks(rotation=90)

plt.title('Distribution of audio frames, per label', fontsize=16)

plt.show()



We observe:

1. The distribution of audio length across labels is non-uniform and has high variance.

Let's now analyze the frame length distribution in Train and Test.

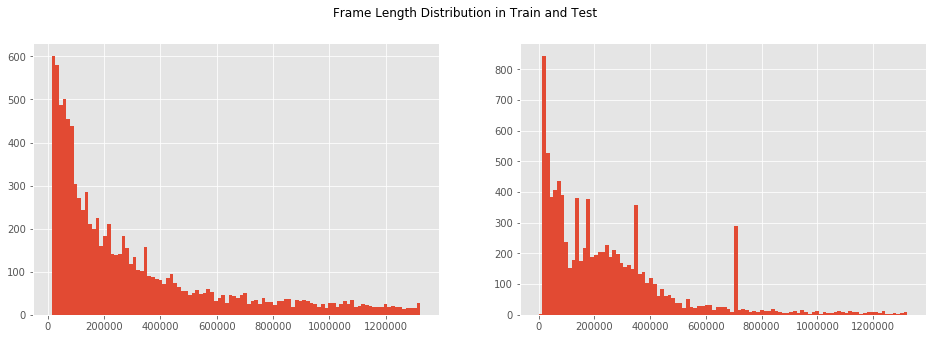
In [15]:

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(16,5))

train.nframes.hist(bins=100, ax=axes[0])

test.nframes.hist(bins=100, ax=axes[1])

plt.suptitle('Frame Length Distribution in Train and Test', ha='center', fontsize='large');



We observe:

1. Majority of the audio files are short.
2. There are four abnormal length in the test histogram. Let's analyze them.

In [16]:

abnormal\_length = [707364, 353682, 138474, 184338]

for length in abnormal\_length:

abnormal\_fnames = test.loc[test.nframes == length, 'fname'].values

print("Frame length = ", length, " Number of files = ", abnormal\_fnames.shape[0], end=" ")

fname = np.random.choice(abnormal\_fnames)

print("Playing ", fname)

IPython.display.display(ipd.Audio( '../input/freesound-audio-tagging/audio\_test/' + fname))

Frame length = 707364 Number of files = 210 Playing 87f52da2.wav

Frame length = 353682 Number of files = 127 Playing 1819a7b7.wav

Frame length = 138474 Number of files = 170 Playing 32bdd578.wav

Frame length = 184338 Number of files = 153 Playing 113d2ca9.wav

## 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:

### 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 esssential imports

Output

In [17]:

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, ReduceLROnPlateau)

from keras.layers import (Convolution1D, Dense, Dropout, GlobalAveragePooling1D,

GlobalMaxPool1D, Input, MaxPool1D, concatenate)

from keras.utils import Sequence, to\_categorical

#### 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 [18]:

class Config(object):

def \_\_init\_\_(self,

sampling\_rate=16000, audio\_duration=2, n\_classes=41,

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/512)), 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 [19]:

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\_size]

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\_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")

# Normalization + Other Preprocessing

if self.config.use\_mfcc:

data = librosa.feature.mfcc(data, sr=self.config.sampling\_rate,

n\_mfcc=self.config.n\_mfcc)

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:

return X

#### Normalization

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

In [20]:

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 [21]:

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, metrics=['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, metrics=['acc'])

return model

#### Training 1D Conv

It is important to convert raw labels to integer indices

In [22]:

LABELS = list(train.label.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.label.apply(lambda x: label\_idx[x])

if not COMPLETE\_RUN:

train = train[:2000]

test = test[:2000]

In [23]:

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\_epochs=1)

Here is the code for 10-fold training:

* We use from sklearn.cross\_validation.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.

Output

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(train.label\_idx, n\_folds=config.n\_folds)

for i, (train\_split, val\_split) in enumerate(skf):

train\_set = train.iloc[train\_split]

val\_set = train.iloc[val\_split]

checkpoint = ModelCheckpoint('best\_%d.h5'%i, monitor='val\_loss', verbose=1, save\_best\_only=True)

early = EarlyStopping(monitor="val\_loss", mode="min", patience=5)

tb = TensorBoard(log\_dir='./logs/' + PREDICTION\_FOLDER + '/fold\_%d'%i, write\_graph=True)

callbacks\_list = [checkpoint, early, tb]

print("Fold: ", i)

print("#"\*50)

if COMPLETE\_RUN:

model = get\_1d\_conv\_model(config)

else:

model = get\_1d\_dummy\_model(config)

train\_generator = DataGenerator(config, '../input/freesound-audio-tagging/audio\_train/', train\_set.index,

train\_set.label\_idx, batch\_size=64,

preprocessing\_fn=audio\_norm)

val\_generator = DataGenerator(config, '../input/freesound-audio-tagging/audio\_train/', val\_set.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\_multiprocessing=True, workers=6, max\_queue\_size=20)

model.load\_weights('best\_%d.h5'%i)

# Save train predictions

train\_generator = DataGenerator(config, '../input/freesound-audio-tagging/audio\_train/', train.index, batch\_size=128,

preprocessing\_fn=audio\_norm)

predictions = model.predict\_generator(train\_generator, use\_multiprocessing=True,

workers=6, max\_queue\_size=20, verbose=1)

np.save(PREDICTION\_FOLDER + "/train\_predictions\_%d.npy"%i, predictions)

# Save test predictions

test\_generator = DataGenerator(config, '../input/freesound-audio-tagging/audio\_test/', test.index, batch\_size=128,

preprocessing\_fn=audio\_norm)

predictions = model.predict\_generator(test\_generator, use\_multiprocessing=True,

workers=6, 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)

#### Ensembling 1D Conv Predictions

Now that we have trained our model, it is time average the predictions of 10-folds. We will try Geometric Mean averaging and see what will be our Public LB score.

Output

In [24]:

pred\_list = []

for i in range(10):

pred\_list.append(np.load("../input/freesound-prediction-file/test\_predictions\_%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/freesound-audio-tagging/sample\_submission.csv')

test['label'] = predicted\_labels

test[['fname', 'label']].to\_csv("1d\_conv\_ensembled\_submission.csv", index=False)

## 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/).

#### 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 [25]:

import librosa

SAMPLE\_RATE = 44100

fname = '../input/freesound-audio-tagging/audio\_train/' + '00044347.wav' # Hi-hat

wav, \_ = librosa.core.load(fname, sr=SAMPLE\_RATE)

wav = wav[:2\*44100]

Output

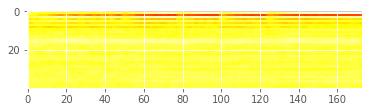
In [26]:

mfcc = librosa.feature.mfcc(wav, sr = SAMPLE\_RATE, n\_mfcc=40)

mfcc.shape

In [27]:

plt.imshow(mfcc, cmap='hot', interpolation='nearest');



## 4. Building a Model using MFCC

We will build now build a 2D Convolutional model using MFCC.

Output

In [28]:

from keras.layers import (Convolution2D, GlobalAveragePooling2D, BatchNormalization, Flatten,

GlobalMaxPool2D, MaxPool2D, concatenate, Activation)

from keras.utils import Sequence, to\_categorical

from keras import backend as K

In [29]:

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, metrics=['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, metrics=['acc'])

return model

### Preparing data

In [30]:

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 [31]:

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=config.n\_mfcc)

data = np.expand\_dims(data, axis=-1)

X[i,] = data

return X

X\_train = prepare\_data(train, config, '../input/freesound-audio-tagging/audio\_train/')

X\_test = prepare\_data(test, config, '../input/freesound-audio-tagging/audio\_test/')

y\_train = to\_categorical(train.label\_idx, num\_classes=config.n\_classes)

#### Normalization

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

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(train.label\_idx, n\_folds=config.n\_folds)

for i, (train\_split, val\_split) in enumerate(skf):

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', verbose=1, save\_best\_only=True)

early = EarlyStopping(monitor="val\_loss", mode="min", patience=5)

tb = TensorBoard(log\_dir='./logs/' + PREDICTION\_FOLDER + '/fold\_%i'%i, write\_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=callbacks\_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)

#### Ensembling 2D Conv Predictions

In [32]:

pred\_list = []

for i in range(10):

pred\_list.append(np.load("../input/freesound-prediction-data-2d-conv-reduced-lr/test\_predictions\_%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/freesound-audio-tagging/sample\_submission.csv')

test['label'] = predicted\_labels

test[['fname', 'label']].to\_csv("2d\_conv\_ensembled\_submission.csv", index=False)

## 5. Ensembling 1D Conv and 2D Conv Predictions

In [33]:

pred\_list = []

for i in range(10):

pred\_list.append(np.load("../input/freesound-prediction-data-2d-conv-reduced-lr/test\_predictions\_%d.npy"%i))

for i in range(10):

pred\_list.append(np.load("../input/freesound-prediction-file/test\_predictions\_%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/freesound-audio-tagging/sample\_submission.csv')

test['label'] = predicted\_labels

test[['fname', 'label']].to\_csv("1d\_2d\_ensembled\_submission.csv", index=False)

## Results and Conclusion

So far, we have trained two models. Let's analyze their relative complexity and strength.

| Model | Number of Trainable parameters | Public LB score |
| --- | --- | --- |
| 1D Conv on Raw wave | 360,513 | 0.809 |
| 2D Conv on MFCC (verified labels only) | 168,361 | 0.785 |
| 2D Conv on MFCC | 168,361 | 0.844 |
| 1D Conv + 2D Conv Ensemble | N/A | 0.895 |

As we can see, 2D Convolution on MFCC performs better than 1D Convolution on Raw waves.

**Coming Soon**

1. Data Augmentation
2. Training on Manually Verified Labels

This kernel has been released under the [Apache 2.0](http://www.apache.org/licenses/LICENSE-2.0) open source license.